DOCUMENT RESUME

ED 377 819 IR 016 915

AUTHOR	Frazier, Michael Duane
TITLE	Matters Horn and Other Features in the Computational Learning Theory Landscape: The Notion of Membership.
INSTITUTION	Illinois Univ., Urbana. Dept. of Computer Science.
REPORT NO	UILU-ENG-94-1716; UIUCDCS-R-94-1858
PUB DATE	Apr 94
NOTE	188p.; Ph.D. Dissertation, University of Illinois at Urbana-Champaign.
PUB TYPE	Dissertations/Theses - Doctoral Dissertations (041) -- Reports - Evaluative/Feasibility (142)
EDRS PRICE	MF01/PC08 Plus Postage.
DESCRIPTORS	Algorithms; *Automation; *Coding; Computation; Data Collection; *Group Membership; *Learning Theories; Problem Solving
IDENTIFIERS	*Computational Learning Theory; *Horn Sentences; Knowledge Acquisition; Representation Language; Uncertainty

ABSTRACT

Computer task automation is part of the natural progression of encoding information. This thesis considers the automation process to be a question of whether it is possible to automatically learn the encoding based on the behavior of the system to be described. A variety of representation languages are considered, as are means for the learner to acquire a variety of types of data about the system in question. The learning process is abstracted as a learning problem in which the goal is to efficiently collect sufficient information to identify some hidden concept using a particular language. The source of information about the concept is its relationship to some class of examples that is assumed to be reasonably available even if the concept is not. The goal of inquiry is to produce a learning algorithm that automates encoding of any representation (or to show that none is possible). It is argued that learning algorithms exist for two natural representation languages: propositional Horn sentences and the CLASSIC description logic, a natural first-order class used in the knowledge representation community. A new method is introduced for modeling uncertainty in the information being collected. Tools that have been developed in computational learning theory can be used for automation in real world tasks outside learning theory. Twenty-two figures are included. (Contains 101 references.) (Author/SLD)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it

☐ Minor changes have been made to improve reproduction quality

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

ED 377 819

Department of Computer Science
University of Illinois at Urbana-Champaign

Technical Report

Leading the world in computing since 1948.

REPORT NO. UIUCDCS-R-94-1858 UILU-ENG-94-1716

Matters Horn and Other Features in the Computational Learning Theory Landscape: The Notion of Membership

by

Michael Duane Frazier

April 1994

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

J.L. Pence

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR016915

REPORT NO. UIUCDCS-R-94-1858

Matters Horn and Other Features in the Computational Learning Theory Landscape: The Notion of Membership

by

Michael Duane Frazier

April 1994

DEPARTMENT OF COMPUTER SCIENCE
1304 W. SPRINGFIELD AVENUE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, IL 61801

©Copyright by
Michael Duane Frazier
1994

MATTERS HORN AND OTHER FEATURES IN THE COMPUTATIONAL LEARNING THEORY LANDSCAPE: THE NOTION OF MEMBERSHIP

BY

MICHAEL DUANE FRAZIER

B.S., University of Missouri – Rolla, 1985
M.S., University of Missouri – Rolla, 1987

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1994

Urbana, Illinois

Computer automation of tasks is part of the natural progression of encoding information. When the task becomes well understood and repetitive, placing the task under computer control becomes a possibility. Computers were once programmed by rewiring rather than with the use of a modern program, management of a limited memory was once handled by the application programmer rather than by the operating system, and efficient use of the computer's hardware was once obtained by assembly language programmers rather than through a compiler. In other areas, accounting moved from ledger books to spreadsheets, automobile fuel intake left the carburetor for computer-controlled fuel injection, and diagnosis and scheduling left the expert for the expert system.

Current knowledge representation research has sought to provide schemes for encoding knowledge about how a given system behaves, with the goal being accuracy and utility. Can an accurate description be given with the representation language being used? Can the resulting representation be manipulated easily to answer questions about the system being described? To the extent that both questions can be answered affirmatively for some representation language $\mathcal{L}$, encoding information using $\mathcal{L}$ is well understood. Ideally, the goal of encoding knowledge is not the task of encoding, but the product of the encoding task. If such encodings are required for a variety of systems, then question of automating the process of encoding arises.

This thesis considers this automation process to be a question of whether it is possible to automatically learn the encoding based on the behavior of the system to be described. A variety of representation languages $\mathcal{L}$ are considered, as are a variety of means for the learner to acquire a variety of types of data about the system in question. The learning process is abstracted as a *learning problem* in which the goal is to collect efficiently sufficient information to identify some hidden *concept* C represented using the language $\mathcal{L}$. The source of information about C is its relationship to some class of *examples* $\mathcal{X}$ that is assumed to be reasonably available even though C itself is not. In addition to conjecturing guesses as to the identity of C, the learner is permitted ask how C relates to individuals $x \in \mathcal{X}$.

The goal of inquiry about this automation process is either to produce a learning algorithm that efficiently automates the encoding of *any* representation that uses some useful representation language $\mathcal{L}$ or to show that no such learning algorithm is possible. The centerpiece of this thesis is that there do exist learning algorithms for two natural representation languages: propositional Horn sentences and the CLASSIC description logic. In addition, this thesis introduces a new method – *consistently ignorant teachers* – of modeling uncertainty in the information being collected. The goal of this thesis is to demonstrate that, by careful consideration of the task at hand, the tools that have been developed in the field of computational learning theory can be used to automate the process of constructing the explanations required by real-world tasks in fields outside computational learning theory.

In all the arts & crafts classes I took as a child, I do not remember ever making a drink coaster for my parents; my hope is that this pile of paper will suffice. Mention also goes to my second grade teacher, Mrs. Farmer, who apparently was more confident of my completion of this stage of education than I.

Most importantly, however, I dedicate this work to my loving wife, Mary, who has been a great source of encouragement to complete this thesis so that we may truly begin our new life together.

ACKNOWLEDGEMENTS

I wish to express appreciation for the members of my committee – Professors Alan Frisch, Sally Goldman, Michael Loui, Lenny Pitt, and Ed Reingold – all of whom in one way or another lent me their time and encouragement.

Personally, I owe many thanks to my advisor, Lenny Pitt, who seemed always to have an office full of toys and motivation to spare, to David Page who was equally open to discussions of Christianity or inductive logic programming, and to Dana Angluin and Sally Goldman for their clarity of prose. Thanks also go to Nina Mishra and Howard Aizenstein for uncounted discussions when my insight departed. I also wish to acknowledge the invaluable support of NASA grant NAG 1-613, NSF grant IRI-9014840, the Department of Computer Science, and the Beckman Institute.

Regarding specific contributions, I gratefully acknowledge the following. Thanks goes to Dana Angluin, Alan Frisch, and Michael Kearns for valuable discussions concerning the work in chapter 3 on entailed Horn learning. Thanks also goes to Dana Angluin for helpful conversations regarding the work in chapter 5 on consistently ignorant teachers. Finally, thanks goes to William Cohen and Haym Hirsh for invaluable discussions and clarifications in chapter 7 regarding the CLASSIC language.

PREFACE

True wisdom lies not in possessing knowledge but in sharing it.

TABLE OF CONTENTS

Chapter

1 Introduction to Computational Learning Theory 1
- 1.1 State of the Art 5
- 1.2 Tools of the Trade 8
  - 1.2.1 Hardness Results 8
  - 1.2.2 Learnability Results and Prediction 8
  - 1.2.3 Chernoff Bounds 11
  - 1.2.4 Importance of Membership Queries 12
- 1.3 Content 12

2 Propositional Horn Sentences and Membership by Satisfaction 15
- 2.1 The Problem 16
- 2.2 Preliminaries 18
- 2.3 The Algorithm 20
- 2.4 Correctness and Running Time 27
- 2.5 Improvements to the Algorithm 30
- 2.6 Compression 35
- 2.7 Hardness Results 36
- 2.8 Discussion 38

3 Propositional Horn Sentences and Membership by Logical Entailment . . 40
- 3.1 Introduction 42
  - 3.1.1 Approximate Entailment 43
- 3.2 Related Work 44
- 3.3 The Algorithm 46
  - 3.3.1 Learning by Reduction 46
  - 3.3.2 Learning Directly 49
  - 3.3.3 Membership Queries are Necessary 56
- 3.4 Application to Approximate Entailment 58
- 3.5 Discussion 60

4 Membership by Subsumption 62
- 4.1 Definitions 63
- 4.2 The GENERIC Algorithm 65
  - 4.2.1 A Note on Disposing of Equivalence Queries 66
- 4.3 A Newly Learnable Class 68

4.4 Queries About Proof Length . . . 69
4.5 Discussion . . . 69

5 Consistently Ignorant Teachers: Membership by Consensus . . . 71
5.1 Background and Related Work . . . 73
5.2 The Model of a Consistently Ignorant Teacher . . . 74
5.3 An Alternate Formulation of The Model . . . 75
5.4 Positive Results for Learning Agreements . . . 77
5.4.1 Learning Agreements of Nested Concepts . . . 78
5.4.2 A General Technique for Learning Agreements . . . 80
5.4.3 Learning Unions of Boxes in Euclidean Space . . . 82
5.5 A Negative Result . . . 93
5.6 Relating Agreements and Version Spaces . . . 96
5.7 Discussion . . . 99

6 Restricted First-Order Horn Sentences . . . 101
6.1 The Model . . . 102
6.2 The Class $\mathcal{H}_{2,1}$ . . . 103
6.2.1 $\beta<\alpha$ and $\beta<\gamma$ . . . 105
6.2.2 $\alpha<\beta$ and $\beta<\gamma$ . . . 106
6.2.3 $\alpha<\gamma$ and $\gamma<\beta$ . . . 107
6.2.4 $\alpha<\beta$ and $\gamma<\alpha$ . . . 108
6.2.5 $\beta<\alpha$ and $\gamma<\beta$. . . . 108
6.3 Increasing Predicate Arity . . . 110
6.4 Increasing Function Arity . . . 118
6.5 Increasing the Number of Clauses: The Class $\mathcal{H}_{*,1}$ . . . 120
6.6 The Regular Languages Contain the Prefix Languages . . . 122
6.6.1 Prefix Grammar to Right-Linear Grammar Transformation Algorithm . . . 122
6.7 The Prefix Languages Contain the Regular Languages . . . 126
6.8 Applications to Learnability . . . 128
6.9 Related Work . . . 128

7 Description Logics . . . 130
7.1 Introduction . . . 130
7.1.1 CLASSIC . . . 131
7.1.2 The Learning Problem . . . 133
7.1.3 Comparison to Previous Results . . . 135
7.2 The Algorithm . . . 138
7.3 Equivalence Graphs . . . 139
7.4 Labeled Equivalence Graphs . . . 144
7.5 Application to CLASSIC . . . 147
7.6 The Insufficiency of Membership Queries . . . 153
7.7 Membership Query Response Errors . . . 155
7.8 Summary . . . 158

8 Conclusion 159

Appendix

A Thesis Synopsis 160

Bibliography 163

Vita 173

LIST OF FIGURES

1.1 DFA schema for testing example form. 10

2.1 Algorithm for Learning Horn Sentences. 25
2.2 New Version of Algorithm for Learning Horn Sentences. 34

3.1 Membership oracle for HORN. 47
3.2 Equivalence oracle for HORN. 47
3.3 The algorithm entailHORN used to learn Horn sentences. 51
3.4 Find all consequents of entailed clauses having a given antecedent. 51

4.1 The GENERIC learning algorithm. 67

5.1 A method for learning the agreement of nested concepts. 79
5.2 Algorithm to learn a union of origin-incident boxes. 85
5.3 Decomposition of an axis-parallel box with respect to the intersection region. . . 88
5.4 Sub-region constraints imposed by the dimension of the boundary shared with the intersection region. 89
5.5 An example assigned to the wrong sub-region. 90
5.6 Algorithm to expand an underestimate A of the the true intersection region A^* to an estimate A' such that for any point p in S, the sub-region generated by A' in which p lies is the same as the sub-region generated by A^* in which p lies. . . 91
5.7 The algorithm LearnBoxesAgreement for learning the agreement of a set of axis-parallel boxes with samplable intersection region. 92

7.1 A labeled equivalence graph. 136
7.2 The universally positive example for equivalence graphs. Σ indicates that every possible edge label $\sigma \in \Sigma$ appears on a directed edge from the root vertex back to itself. 137
7.3 Equivalence graphs learning algorithm. 138
7.4 Algorithm using membership queries to remove excess graph elements from a positive example. 139
7.5 Updated **Prune**. 152
7.6 A target schema requiring exponentially many membership queries. 153
7.7 This schema requires exponentially many membership queries even though Σ is known to be the set $\{\sigma_1, \sigma_2\}$. 155

Chapter 1

Introduction to Computational Learning Theory

Computational learning theory formalizes the notion of inferring a concept that correctly explains observed data. As such there must be a formalization of "concept," "correct," and "data". These ideas are best formalized simultaneously.

Data exists in units called *examples*. A *concept* is a classifier; it divides the world of examples into (generally) two groups – *positive examples* that exemplify the concept, and *negative examples* that do not exemplify the concept. Thus, a concept can be thought of as a boolean function to *label* examples as either positive or negative.

To illustrate, we might take "duck," "giraffe," "elephant," "table," and "hubcap" as examples. The concept *name of an animal* labels the first three as positive examples and the last two as negative examples; on the other hand, the concept *two syllable names* labels the second, fourth, and fifth as positive and the rest negative.

In a learning problem we assume that a concept has been chosen and fixed, and we must deduce the concept based solely on the labels of the examples. Continuing the illustration, given that "duck" and "table" are the only positive examples in the list, after a moment of thought we might deduce that *verb* is the concept. Unfortunately, we might also deduce that *either "duck" or "table"* is the concept.

This last observation brings us to the formalization of "correct." It is not enough for a deduced concept to correctly label the list of known examples, it must also accurately predict

the label of examples that are as yet unseen. Thus although *verb* and *either "duck" or "table"* are identical in terms of the examples we know about, they are quite different faced with the new example "run."

To summarize, the learning problem assumes that some hidden concept is chosen and the labels for the examples are assigned according to that hidden concept. The learner is expected to deduce from a small set of examples a concept that accurately predicts the label of *every* example (both seen and unseen), and the learner is expected to accomplish this task efficiently. In order to limit the possible explanations, we further assume that the hidden concept was selected from a set of concepts, called the *concept class*, that is known to the learner. We begin with the following formal learnability definition due to Angluin [4].

Definition 1 (Exact Learnability) *Let $\mathcal{X}$ be a set of* examples, *let $\mathcal{C}$ denote a* concept class *consisting of* concepts *expressed in some representation language $\mathcal{L}$ that are total boolean valued functions over the domain $\mathcal{X}$. For any $C \in \mathcal{C}$ let* size(C) *denote the number of symbols needed to represent C using $\mathcal{L}$. Let $\mathcal{A}$ be an algorithm designed with full knowledge of the preceding items. Then $\mathcal{A}$ is an* exact learning algorithm *if there exists a polynomial p such that for any choice of* target $C_* \in \mathcal{C}$ *(with C_* unknown to $\mathcal{A}$), $\mathcal{A}$ outputs in time $p(size(C_*))$ a concept $C'_* \in \mathcal{C}$ that is functionally equivalent to C_* and makes at most $p(size(C_*))$* equivalence queries, *where an equivalence query is made by $\mathcal{A}$ by selecting some $C \in \mathcal{C}$ and then being told that C is functionally equivalent to C_* or being provided with some* counterexample $x \in \mathcal{X}$ *such that $C_*(x) \neq C(x)$. If there exists such an $\mathcal{A}$, the concept class $\mathcal{C}$ is said to be* exactly learnable.

Another second learnability definition, due to Littlestone [72], captures the notion of ongoing learning that eventually produces a correct concept but has made few mistakes along the way. Hence this is sometimes called *mistake bounded learning* or *on-line learning.*

Definition 2 (On-line Learnability) *If there exists a polynomial p such that for any choice of $C_* \in \mathcal{C}$ and any (perhaps infinite) sequence S of examples, $\mathcal{A}$ is called an* on-line learning algorithm *if $\mathcal{A}$, when presented the examples x_i in S, predicts the label of each x_i knowing only $x_1, \ldots, x_{i-1}$ and their correct labeling according to C_*, mispredicts at most $p(size(C_*))$ of the examples in the sequence, and takes time at most $p(size(C_*))$ correcting for a misprediction. If such an $\mathcal{A}$ exists, the concept class $\mathcal{C}$ is said to be* on-line learnable.

When the *evaluation problem* (i.e. computing the value of $C(x)$ for any $C \in \mathcal{C}$ and any $x \in \mathcal{X}$) is solvable in polynomial time, it is easy to see that an on-line learning algorithm that keeps a current $C \in \mathcal{C}$ to use in predicting the label of the next example in Littlestone's setting exists if and only if an exact learning algorithm exists; the equivalence query essentially forces the next prediction error to be presented immediately whereas the sequence S allows the C used in the exact learning algorithm's equivalence query to be profitably used until the need arises for correction [71].

A probabilistic variation on the definition of learnability is due to Valiant [98].

Definition 3 (PAC Learnability) *If there exists a polynomial p such that for any choice of $C_* \in \mathcal{C}$, for any probability distribution D over $\mathcal{X}$, and given any $\epsilon > 0$ and $\delta > 0$, $\mathcal{A}$ is called a* probably approximately correct (PAC) learning algorithm *if $\mathcal{A}$ sees at most $p(size(C_*), 1/\epsilon, 1/\delta)$ examples selected according to D and outputs in time $p(size(C_*), 1/\epsilon, 1/\delta)$ a concept C'_* such that with probability at least $1 - \delta$, the probability that $C_*(x) \neq C'_*(x)$ on a point x chosen randomly according to D is at most ϵ with respect to D. If such an $\mathcal{A}$ exists, the concept class $\mathcal{C}$ is said to be* PAC learnable.

Littlestone [72] shows that a PAC learning algorithm for a given concept class exists if an on-line learning algorithm for that class exists. The idea used is that the on-line learning algorithm tests its current hypothesis by taking a polynomial number of random examples. If the hypothesis has significant error, one of the random examples chosen will demonstrate this error and can be used by the on-line algorithm as a misprediction. The on-line algorithm is guaranteed to make only a polynomial number of mistakes, so that the result is that throughout the run, at most a polynomial number of random examples are witnessed before, with high confidence, an accurate hypothesis is produced.

However, for concepts over examples from a continuous domain, PAC learnability does not imply exact learnability; the concept class $\{[0, x) : 0 < x \leq 1\}$ is PAC learnable but not exactly learnable. Even in a discrete domain Blum [19], assuming the existence of one-way functions, gives a hardness result for on-line learnability (and thus exact learnability as well) for a PAC learnable class of functions. Thus, assuming that one-way functions exist, a positive learning result in the exact model is stronger than a positive learning result in the PAC model for boolean concept classes in which the evaluation problem is solvable in polynomial time.

These definitions implicitly assume the size of the representations of the examples is insignificant compared to the size of the representation of the target concept. This is a valid assumption for many problems in computational learning theory. The concept class is frequently some set of boolean formulas, and the examples are frequently bit vectors representing variable settings. For problems in which this assumption is not valid, such as when the concept class is the set of deterministic finite automata and the examples are strings labeled according to whether they are in the language accepted by the target DFA, the bound on the running time of $\mathcal{A}$ is often allowed to depend at most polynomially in the length of the longest example seen. Care must be exercised in these settings to prevent a clever $\mathcal{A}$ from forcing an exponentially longer counterexample to be given in order to justify after the fact the amount of computation time used in identifying the target [3]. In these settings, $\mathcal{A}$ is required at each step in its run to have used no more than time polynomial in the other parameters and the length of the longest counterexample seen to that point in the run.[1]

We will make a number of modifications to the above definitions to adapt them to a variety of settings. Among these modifications will be adding queries of the sort suggested by Angluin [4]; most important among these queries for our purposes is the *membership query* in which $\mathcal{A}$ is permitted to ask at most polynomially many questions of the form "What is $C_*(x)$?" for any $x \in \mathcal{X}$. We will often speak of the randomly chosen examples or answers to queries as being provided by a teacher, an expert, or nature. More pragmatically, these queries may be considered experiments designed by the learner.

The overriding goal of computational learning theory is to devise efficient learners for natural concept classes or to show that no such learner exists. Natural concept classes are taken to be those coming into existence outside the field of computational learning theory because having arisen outside the field suggests that the concept class has interest apart from the question of its learnability. For example, the field of knowledge representation has produced a number of languages and systems [25, 28, 39, 77, 17], each developed to model phenomena from some domain. To the extent that such knowledge representation languages can accurately encode an explanation of the phenomena for which they were intended, the question naturally arises,

[1] In the statement of our results the phrase "time polynomial in the length of the longest counterexample" should be taken to mean that at each step in the run of the algorithm, at most time polynomial in the length of the longest counterexample seen to that point has been used.

"Can the encoding of the explanation be automated, i.e. learned, simply through interaction with phenomenon?" Generally this answer is "No;" the richness of expression within these natural concept classes thwarts known learning attacks. Because of this, approximations are made to the natural concept classes and the learnability of these approximations are considered. Even so, this thesis does devise exact learners for two natural kinds of classifiers – propositional Horn sentences (a standard choice for encoding expert system knowledge) and the description logic CLASSIC (one of a collection of description logics finding many knowledge-based applications [25]), the former using variable assignments as examples and the latter using other CLASSIC descriptions as examples.

We begin by summarizing results relevant to the new work presented here and then discuss some tools used to achieve these new results.

1.1 State of the Art

What natural classes of formulas are learnable? The most natural class of propositional boolean formulas, namely, the class of all boolean formulas, was shown not to be learnable given cryptographic evidence by Kearns and Valiant [66]. Carefully note that this negative result relies on the fact that no restriction was placed on the boolean formula representing the target boolean function. However, if we demand that the target boolean function be represented as, say, a k-CNF formulas (conjunctions of at most k-literal disjuncts, for some constant k), then learning algorithms exist [98, 4]. The discrepancy is that some boolean functions are represented only by exponentially long k-CNF formulas so that the learning algorithm gets more time to learn the same function. Thus, it is not necessarily the class of functions that is hard to learn, but it can be simply that the class of formulas permitted to represent those functions are hard to learn. In other words, the choice of representation automatically provides a complexity parameter which may offer the learner more time than some other representation for the same function.

The learnability question for two other natural classes, namely, CNF and DNF, was left open by Valiant [98] when he introduced the distribution-free or PAC criterion for concept learning. The learnability of these two classes remains open, and efforts to close these questions have led researchers to investigate the learnability of a variety of restricted classes of boolean formulas.

Algorithms exist[2] for learning monomials (pure conjunctive concepts) [73, 98], internal disjunctive concepts [54], read-once formulas [9], monotone DNF formulas [4, 98], k-CNF and k-DNF formulas for constant k [23, 73, 99], and k-term functions also for constant k [22]. There are also algorithms for linearly separable formulas [73], decision lists [88], rank k decision trees (k constant) [42], and decision trees [29], among others. Thus, a number of results have been obtained for approximations to a natural class of formulas. One key result in this thesis is a learning algorithm for a natural class of formulas, propositional Horn sentences, whose learnability was left open by Angluin [2] when she presented a learning algorithm for the approximating class of acyclic Horn sentences.

A great deal of work in the artificial intelligence and knowledge representation communities deals not with propositional but with first-order concepts; as such, there is a wealth of potential in studying the learnability of first-order concepts. Relatively little work has been done within this framework, though interest is rising sharply [70, 81, 82, 15]. Also, Džeroski et al. [41] describe an algorithm that learns k-clause, determinate, function-free, first-order Horn clauses with bounded depth variables by transforming the target into a propositional monotone k-term DNF formula. This thesis gives a learning algorithm for another very restricted subclass of recursive first-order Horn sentences. The primary distinction of this work from that of Džeroski et al. [41] is that the class studied is not *determinate* because functions can be nested to arbitrary depth.[3]

First order learning results have been hard to achieve; to date, the positive learning results for first-order concepts have been for very restricted subclasses of first-order logic. For example, Page and Frisch [82] looked at atoms labeled according to whether they are entailed by a particular kind of hidden first order formula. Other related first-order results come from the field of inductive logic programming where the target is a Prolog program. Cohen [35] gives a PAC learning algorithm for function-free, two clause, linearly recursive, closed, ij-determinant logic programs against a given background theory. Shapiro [95] describes a system for learning Prolog programs *in the limit* (ie, exact learning, but time is an unbounded resource) using

[2]The results use a number of different protocols; in particular some of these results use membership queries. The protocol used to achieve a result stated here is explained in more detail when it becomes relevant later in the thesis.

[3]More precisely, the classes that result from *flattening* (rewriting to contain no functions) the concepts are not determinate.

atoms entailed by the program to be learned. Another key result of this thesis is a learning algorithm for a natural first-order class, CLASSIC, which is used in the knowledge representation community. This learning result is inspired by other work on CLASSIC [36, 33].

An interesting question arises when changing the setting of a learning problem from the propositional domain to the first-order domain: How are examples represented? In the propositional case there is a natural choice for representing examples – bit vectors specifying variable settings. In the first order case, the models over which the semantics of first-order logic are defined are frequently infinite, rendering the phrase "polynomial running time" for an algorithm seeing such examples meaningless. Instead of models labeled according to whether the target formula is satisfied, logical formulas labeled according to entailment by the target are often used. This choice of examples used in learning some particular, unknown body of knowledge suits the artificial intelligence and knowledge representation settings well in that the questions normally asked within those communities involve what logical implications can be efficiently deduced from a particular knowledge base. A third contribution of this thesis arises from carrying the notion of entailment to the propositional domain to provide new results involving examples that are not bit vectors but propositional formulas. Relevant work here comes from the knowledge compilation literature [37, 52, 63, 92]. The results in this thesis include some sufficient conditions for learnability under entailment.

Finally, for when the world does not operate in a well behaved way, such as when questions are answered not by an omniscient source but by a fallible rational source, using a representation closely related to Mitchell's version spaces [79], this thesis provides a model, called a *consistently ignorant teacher*, for the resultant uncertainty and considers a number of learning problems in such a setting.

Interspersed among the positive learning results are a number of non-learnability results, many of which show that some learnable class cannot be extended in a particular way while retaining learnability according to the same criteria. Other non-learnability results show that the types of queries used by a learning algorithm form a minimal set of queries needed to achieve learnability for the class. An itemized list of the contributions made by this thesis appears in the appendix.

1.2 Tools of the Trade

1.2.1 Hardness Results

For relating the difficulty of predicting classes of formulas – and thus for obtaining non-learnability results assuming the hardness of learning some class of formulas – Pitt and Warmuth [86] provide the powerful method known as prediction-preserving reductions. For example, one of the reductions given by Kearns, Li, Pitt and Valiant [65] shows that PAC learning monotone CNF and DNF formulas (without membership queries) is as hard, modulo a polynomial time transformation, as PAC learning general CNF and DNF formulas (without membership queries). Some of the negative results claimed in this thesis were obtained using this method, so those negative results assume that learning CNF and DNF formulas is hard; in the event that learning algorithms do exist for these presumably negative results, the reductions given provide learning algorithms for CNF and DNF, something of great value indeed. At present, however, the learnability of CNF and DNF formulas remains the central open problem in computational learning theory.

Other negative results are based on more common hardness assumptions such as NP $\neq$ RP or the existence of one-way functions. There are also absolute (information theoretic) hardness results that arise from *adversary arguments*; these results are obtained by chosing a target and constructing a set of examples that reveals so little about the target that the learner is forced to request more than a polynomial number of these examples to accurately identify the class.

1.2.2 Learnability Results and Prediction

Reductions can also be used to provide positive learning results. Algorithms can be constructed that provide examples to an existing learning algorithm using a blind transformation of the examples. That is, the learning problem for concept class $\mathcal{C}$ is solved using a learning algorithm for concept class $\mathcal{C}'$ and a polynomial time transformation on examples that does not require knowledge of the chosen target from $\mathcal{C}$. If we require only that the learning algorithm be accurate and not that the algorithm produce a concept from the class $\mathcal{C}$, we can use the concept produced by the learning algorithm for $\mathcal{C}'$ together with the blind transformation on examples as the classifier; the learning algorithm for $\mathcal{C}'$ is then known as a *prediction algorithm* for $\mathcal{C}$.

To illustrate the idea of a prediction algorithm, we present here our first new result. This result extends the work of Blum, Chalasani, and Jackson [20], who describe a learning algorithm for the propositional class of disjoint k-multi-symmetric functions using membership and equivalence queries.

Definition 4 *Let $V = \{x_1, \ldots, x_n\}$ be a set of propositional variables. Then a* disjoint k-multi-symmetric function *is a specification of a boolean valued k-ary function f on tuples of natural numbers together with a specification of k disjoint subsets $S_1, \ldots, S_k$ of V. The value of f is obtained from a setting of the variables of V by evaluating f on the tuple $\langle n_1, \ldots, n_k \rangle$ where n_i is the number of variables in S_i that are set true.*

This definition requires that the k subsets be disjoint. By removing the requirement that an algorithm construct a representation of this form, the class of functions in which these subsets need not be disjoint can be predicted by Angluin's algorithm for learning deterministic finite automata [3]. The construction of an on-line learning algorithm is as follows.

Blum, Chalasani, and Jackson consider f to be represented as a lookup table of $(n+1)^k$ entries, one entry for each possible tuple supplied to f. The value of the function f depends only on the count of the variables of each S_i that are assigned *true*. If we take a variable assignment and write down the variable names in alphabetical order of those variables assigned *true*, then for each i it is an easy task to construct an $n+1$-state DFA whose input symbols are the n variable names and whose states represent the number of variables in S_i assigned *true*. It then follows easily that a DFA having the n variable names as input symbols and having at most $(n+1)^k$ states can be constructed such that the states of the DFA represent the number of variables assigned *true* in any particular k such subsets S_i. By designating as accepting states those states corresponding to those tuples that f labels positive examples, this DFA accepts exactly the variable assignment that f labels positive and the DFA has essentially the same size as the table representing f. We will use this DFA representation of the (not necessarily disjoint) k-multi-symmetric functions, and we will use Angluin's DFA learning algorithm to learn the DFA representation.

We will construct a sequence of examples for the DFA learning algorithm to use from the sequence of examples presented for f. Occasionally, we will insert some of our own examples that serve to constrain the search facing the DFA learning algorithm; it is important to note

that knowledge about f is used neither in the transformation of examples for f into examples for the DFA learning algorithm, nor in the selection of the extra examples inserted into the sequence for the DFA.

Because the examples given to the DFA algorithm are supposed to represent an alphabetized list of variables set *true*, when the DFA learning algorithm wishes to test some DFA A for accuracy, we check to see that the language $L(A)$ it accepts contains only strings in which the symbols appear in alphabetical order and no symbol appears twice. To accomplish this, we make $O(n^2)$ checks that

$$L(A) \cap (V - \{x_j\})^* x_j (V - \{x_i\})^* x_i V^*$$

is the empty language for every pair of symbols x_i and x_j in V with $i \leq j$. Clearly, given x_i and x_j, each of these tests can be performed in polynomial time because each test can be performed by intersecting the 3-state DFA $M(i,j)$ shown in Figure 1.1 with A and checking the resulting language for emptiness. Equally clearly, a string in any of the non-empty languages is efficiently found.

Observe that $M(i,i)$ accepts exactly those strings in which x_i appears more than once, and $M(i,j)$ accepts exactly those strings in which x_j occurs before x_i. If any language $L(A) \cap M(i,j)$ is non-empty, present any string in any of these languages as the next example in the sequence being presented to the DFA learning algorithm.

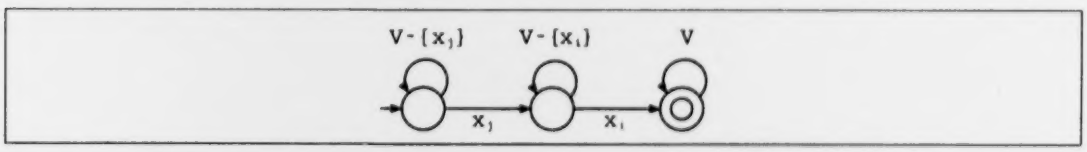

Figure 1.1: DFA schema for testing example form.

If none of these languages is empty, then continue using A against the sequence of examples being presented. For each example, alphabetize the list of variables set *true*, preserve the example label, and hand the result as the next example in the sequence to be presented to the DFA learning algorithm.

The matter is simpler for membership queries. When the DFA learning algorithm poses a membership query, answer the query "no" if the string has any repeated symbols or if the

string is not alphabetized. Otherwise, pose a membership query formed by setting exactly those variables in the string to *true* and respond to the DFA query with the answer received from a variable assignment membership query, which we have at our disposal.

Because Angluin's DFA learning algorithm makes at most a polynomial number of mistakes regardless of the sequence it sees, we have the following result.

Theorem 5 *The class of (not necessarily disjoint) k-multi-symmetric functions is predictable by the on-line version of Angluin's DFA learning algorithm.*

Proof: The above discussion shows that for any (not necessarily disjoint) k-multi-symmetric function f, there is a DFA A_f having size equivalent to the size of the table representation T_f of f. The discussion gives a polynomial time one-to-one mapping s between the set $\mathcal{X}$ of variable assignment examples and the set $\mathcal{X}'$ of alphabetized strings over variable the names in which no symbol occurs twice. It is then shown that T_f labels $e \in \mathcal{X}$ a positive example if and only if $s(e)$ is accepted by A_f. Since the number of prediction mistakes made by the DFA learning algorithm is at most polynomial in the size of A_f, the number of prediction mistakes is also at most polynomial in the size of T_f. Thus the above construction produces an on-line learning algorithm for the class of (not necessarily disjoint) k-multi-symmetric functions. □

1.2.3 Chernoff Bounds

One standard tool used in estimating the likelihood of certain properties of a probability distribution in the PAC setting is Chernoff bounds. One formulation of these bounds, stated below, can be found in [13].

Fact 6 *Let $LE(p, m, (1-\alpha)pm)$ denote the probability that in m independent trials an event having probability of occurrence p is witnessed at most $(1-\alpha)pm$ times, and let $GE(p, m, (1+\alpha)pm$ denote the probability that in m independent trials an event having probability of occurrence p is witnessed at least $(1+\alpha)pm$ times. Then*

- $LE(p, m, (1-\alpha)pm) \leq e^{-\alpha^2 mp/2}$
- $GE(p, m, (1+\alpha)pm) \leq e^{-\alpha^2 mp/3}$

1.2.4 Importance of Membership Queries

Membership queries can make a difference – for example, monotone DNF and CNF formulas are PAC learnable if membership queries are available [98, 4], whereas their status is open without membership queries. Another example is provided by read-once formulas. These are PAC learnable in polynomial time if membership queries are available [9]. However, by another reduction of Kearns, Li, Pitt, and Valiant [65], the PAC learnability of read-once formulas is equivalent to that of general Boolean formulas in the absence of membership queries, and therefore as hard as certain apparently hard cryptographic problems, by the results of Kearns and Valiant [66]. In the exact model, read-once formulas are not learnable with equivalence queries alone [9] bur are exactly learnable with membership queries.

How much are membership queries likely to help with learning general Boolean formulas, or general CNF and DNF formulas? Angluin and Kharitonov [10] give cryptographic evidence that the answer in both cases is "not much." For general boolean formulas there is cryptographic evidence of the same sort as given by Kearns and Valiant that they are not PAC learnable in polynomial time even if membership queries are available. For general CNF and DNF formulas the situation is more complicated, but, in effect, there is cryptographic evidence that either general CNF formulas will be PAC learnable in polynomial time without membership queries, or they won't be PAC learnable in polynomial time even with membership queries – that is, the membership queries "won't help" in the case of general CNF and DNF formulas.

Because the availability of membership queries may change the answer to the learnability question for some learning problems, the results claimed in this thesis include results about the necessity (or lack thereof) of membership queries.

1.3 Content

Each of chapters 2 through 7 of this thesis (except for chapter 4) represents a published work. Chapters 2 through 5 provide propositional results, beginning with the most intuitive model. Chapters 6 and 7 provide first-order results, chapter 7 discussing the less familiar concept class. The discussion of the significance of the results of a particular chapter appear at the end of that chapter.

Chapter 2 represents joint work with Dana Angluin and Lenny Pitt and appears as [8]. This chapter gives the most detailed description of the relationship of variable assignments to propositional formulas. This chapter also carefully defines the class of propositional Horn formulas. This level of detail is omitted in subsequent chapters. The results presented in Chapter 2 include two learning algorithms for the important class of propositional Horn formulas; also presented are hardness results for the related class of 2-quasi Horn sentences. Sections 2.6 and 2.7 includes material left open in [8].

Chapter 3 represents joint work with Lenny Pitt, which appears as [46]. This chapter builds directly on the work in Chapter 2, but trades the familiar variable assignment examples for clauses labeled according to logical entailment by a propositional Horn target. This chapter also presents two learning algorithms for the class of propositional Horn sentences - one a direct learning algorithm, and the other a reduction to either learning algorithm from Chapter 2. This chapter also presents hardness results for attempting to disallow membership queries.

Chapter 4, also representing joint work with Lenny Pitt and also extending the work of Chapter 2, presents a learning protocol motivated by the desire to learn an *efficiently applicable* representation of the target. This desire contrasts with the common desire to learn a *small* representation of the target. The protocol assumes the time taken by an expert to answer a membership query is important, but does not assume that the expert is able to articulate his representation of the target function. Sufficient conditions for learning under this protocol are given and are applied to a class of formulas defined by Boros et al. [27], a class properly containing Horn sentences and 2-CNF.

Chapter 5 represents joint work with Sally Goldman, Nina Mishra, and Lenny Pitt and appears as [44]. A new learning model is presented in this chapter. Here the teacher is assumed to be rational but not necessarily omniscient. Viewed another the way, the teacher may have several competing models of the world and presents categorical answers to the learner only in those cases where the teacher's competing models agree. The learner is expected to acquire the teacher's models of the world. This chapter presents learning algorithms for a variety of concept classes, including a concept class related to the union of boxes in d-dimensional Euclidean space. A hardness result is given for the case in which the teacher has several competing Horn models of the world.

Chapter 6 represents joint work with C. David Page and appears as [43]. This chapter presents a learning algorithm that uses no membership queries, but learns a very restricted first-order class of recursive Horn sentences. Here the learner sees first-order clauses labeled according to entailment by the target. This chapter also presents some discussion on the effect of relaxing some of the restrictions on the formulas in the class; a hardness result is presented when the restriction to unary functions is removed. A secondary result in this chapter is a new characterization of the regular languages.

Chapter 7 represents joint work with Lenny Pitt and appears as [45]. This chapter discusses a surprising positive learnability result for the (first-order) description logic CLASSIC. Here the learner collects examples labeled according to logical consequence of the target. It is shown in this chapter that arbitrary CLASSIC descriptions can be learned, even when the rate of malicious noise in the answer to membership queries is near 1/2.

Chapter 2

Propositional Horn Sentences and Membership by Satisfaction

We (joint work with Angluin and Pitt) show that the class of Horn sentences is exactly learnable using membership and equivalence queries from a standard teacher. This set of queries is minimal in that Angluin [4, 6] has shown that propositional Horn sentences are not exactly learnable from a standard teacher from membership or equivalence queries alone. Observe that Horn sentences are "almost monotone" in the sense that each clause contains at most one positive literal. As a negative result, we show via prediction preserving reduction that the class of 2-quasi Horn sentences – conjunctions of clauses, each clause containing at most two positive literals – is no easier to PAC learn using membership queries than CNF. We also show, using an inciteful observation by Vijay Raghavan [87], that 2-quasi Horn sentences are no easier to exactly learn with equivalence and membership queries from a standard teacher than CNF. Thus in some sense, the amount of "non-monotonicity" permitted in Horn formulas appears to be near the edge of learnability. Further, it is interesting to note the kind of questions the learner is permitted to ask when learning Horn sentences are quite reasonable because the teacher can answer them in polynomial time.

Compared to learning algorithms using membership queries to learn other concept classes, our Horn learning algorithm might be termed "lazy" in that membership queries are commonly aggressively applied to a counterexample to construct a new counterexample that has some minimality property (say, a minimal number of variables assigned *true*). In contrast, our algorithm

works by keeping a sequence S of negative examples, and focuses its attention on explaining why those examples are negative. An example is negative only if it falsifies some clause of the target, and a Horn clause is falsified only if each of the variables in the antecedent of the clause are set to true and the variable in the consequent is set to false. Thus the algorithm explains a negative example in S by assuming the target contains a clause whose antecedent consists of exactly the variables set true by the example and that the consequent (if any) of the class is one the variables the example makes false. Clearly the antecedent might not contain *all* the variables set true by the example, so, in effect, the algorithm waits for another negative example that is explained by the same clause and throws out all the variables these two negative examples do not have in common; further this is the *only* explanation that is updated. This, relaxed, data driven nature of this algorithm seems crucial; a number of more aggressive approaches that attempt to throw out as many variables as possible, or that attempt to update several explanations in response to new information, were foiled when a clever adversary choose the counterexamples.

Because of the nice computational structure of Horn sentences, both the membership *and* the equivalence queries can be answered in polynomial time given knowledge of the target. Thus as a benefit, the learning algorithm can be used to reduce a given Horn sentence to its logical equivalent having the minimum number of distinct antecedents in polynomial time. To our knowledge, there did not exist an efficient procedure for finding a minimum sized Horn sentence equivalent to a given sentence, nor were we aware of canonical forms for Horn sentences. A pleasing property of our algorithm is that it produces guesses as to the identity of the target having a monotonically increasing number of distinct antecedents; thus the algorithm is capable of determining whether the function it is being asked to learn can be represented as a Horn sentence of a given size. This property is useful in trying to approximate a (not necessarily Horn) concept by a small Horn.

2.1 The Problem

Let $V = v_1, \ldots, v_n$ be a set of Boolean variables. A *literal* is either a variable v_i or its negation $\neg v_i$. A *clause* over variable set V is a disjunction of literals. A *Horn clause* is a clause in which at most one literal is unnegated. A *Horn sentence* is a conjunction of Horn clauses. The class

of Horn sentences over variable set V is a proper subclass of the class of Boolean formulas over V.

Let H_* denote the target Horn sentence to be learned. The main result of this chapter is that propositional Horn sentences are exactly learnable using variable assignments as examples, provided membership (and equivalence) queries are available. The algorithm runs in time[1] $\tilde{O}(m^2n^2)$, making $O(mn)$ equivalence queries and $O(m^2n)$ membership queries, where m is the number of clauses, and n is the number of variables of H_*.

It is interesting to note that both types of queries are necessary for learning Horn sentences. Angluin [4] shows that membership queries alone are insufficient for polynomial-time learning, and, implicitly in [6], she proves that equivalence queries alone are also insufficient.

A similar result for learning monotone formulas in disjunctive normal form (DNF) has been given [4, 98]. The dual of the class of Horn sentences is the class of "almost monotone" DNF formulas — a disjunct of terms, where each term is a conjunct of literals, at most one of which is negated. Since our algorithm is easily modified to handle the dual class, it extends the results in [4, 98] by allowing a small amount of nonmonotonicity. (Later, we indicate why allowing more nonmonotonicity would yield a difficult problem.) Horn sentences are an interesting nontrivial subclass of CNF (dual: DNF) formulas, the learnability of which remains a central open problem for the distribution-independent ("PAC learning") model of Valiant [98]. The research presented here also improves the results in [2], where the class of Horn sentences is shown to be learnable by an algorithm that uses equivalence queries that return Horn clauses as counterexamples and "derivation queries" — a type of query that is significantly more powerful than a membership query.

By modifying the algorithm presented here in a relatively straightforward way [4] we could obtain an algorithm that learns the class of Horn sentences using randomly generated examples as in the PAC learning model, provided that the algorithm is additionally allowed to make membership queries. Similarly, the algorithm presented here could be used in an on-line setting in which the learning algorithm is to classify each of a succession of examples, and the algorithm is told whether its classification is correct or incorrect before receiving each next example. The resulting on-line algorithm makes membership queries (excluding the examples to be classified)

[1] The $\tilde{O}()$, or "soft-O", notation is similar to the usual $O()$ notation except that $\tilde{O}()$ ignores logarithmic factors.

but not equivalence queries, and is guaranteed to make at most a polynomial number of errors of classification regardless of the sequence of examples [4].

Note that because the problem of determining whether two Horn sentences are equivalent (and producing a counterexample if they are not) is solvable in polynomial time, the oracle in our learning protocol could be replaced by a teacher with polynomially bounded computational resources.

The remainder of this chapter is organized as follows. Section 2.2 gives basic definitions, notation, and lemmas that will be used throughout. In Section 2.3 we describe the algorithm and give an example run. Section 2.4 provides a correctness proof and an analysis of the time and query complexity of the algorithm. In Section 2.5 we present a modified version of the algorithm which satisfies smaller time and query bounds. We conclude in Section 2.8 by discussing some related and open problems.

2.2 Preliminaries

It is often easier to discuss the satisfaction or falsification of a Horn clause when that clause is represented as an implication. To expedite the discussion we will implicitly assume that all Horn clauses are represented as implications. This necessitates the introduction of two logical constants.

Definition 7 *The logical constant "true" is represented by* **T** *and the logical constant "false" is represented by* **F**.

Next, we introduce notation that will enable us to dissect Horn clauses and discuss the relationships between them and examples. First, recall the identity $z \vee \bigvee_{i=1}^{k} \neg v_i \equiv \left(\bigwedge_{i=1}^{k} v_i\right) \Rightarrow z$, where "$\Rightarrow$" is the logical connective for implication and "$\equiv$" is a metasymbol indicating logical equivalence. Now, taking $\bigwedge_{i=1}^{0} v_i \equiv \mathbf{T}$ (the empty conjunction evaluates to true) and adopting the convention that we write $\left(\bigwedge_{i=1}^{k} v_i\right) \Rightarrow \mathbf{F}$ when there are no unnegated variables in the Horn clause, we have the following definitions.

Definition 8 *Let H be any Horn sentence over V. An* example *is any assignment $x : V \rightarrow \{\mathbf{T}, \mathbf{F}\}$. A* positive *(respectively,* negative*) example for H is an assignment x such that H evaluates to true (respectively, false) when each variable v in H is replaced by $x(v)$.*

Definition 9 *Let x be an example; then $true(x)$ is the set of variables assigned the value "true" by x and $false(x)$ is the set of variables assigned the value "false" by x.*

By convention, $\mathbf{T} \in true(x)$ and $\mathbf{F} \in false(x)$.

Definition 10 *Let C be a Horn clause. Then antecedent(C) is the set of variables that occur negated in C. If C contains an unnegated variable z, then consequent(C) is just z. Otherwise, C contains only negated variables and consequent(C) is $\mathbf{F}$.*

We now describe the relationships that may exist between an example and a Horn clause.

Definition 11 *An example x is said to* cover *a Horn clause C if antecedent(C) $\subseteq true(x)$. We say that x* does not cover *C if antecedent(C) $\not\subseteq true(x)$. The example x is said to* violate *the Horn clause C if x covers C and consequent(C) $\in false(x)$.*

Notice that if x violates C then x must cover C, but that the converse does not necessarily hold.

It will be more convenient throughout the rest of the paper to consider a Horn sentence as a *set* of Horn clauses, representing the conjunction of the clauses.

Our first observation is trivial, but it is helpful to state it formally.

Proposition 1 *If x is a negative example for the Horn sentence H, then x violates some clause of H.*

We next define the $\cap$ operation for examples.

Definition 12 *Let x and s be two examples, then $x \cap s$ is defined to be the example z such that $true(z) = true(x) \cap true(s)$.*

Note that this implies that $false(x \cap s)$ is $false(x) \cup false(s)$.

Lemma 13 *Let x and s be examples. If x violates C and s covers C, then $x \cap s$ violates C.*

Proof: If s covers C then $antecedent(C) \subseteq true(s)$. Also, if x violates C, then $antecedent(C) \subseteq true(x)$ and $consequent(C) \in false(x)$. Thus $antecedent(C) \subseteq true(s) \cap true(x) = true(s \cap x)$ and $consequent(C) \in false(x) \subseteq false(s) \cup false(x) = false(s \cap x)$. Thus, $s \cap x$ violates C. □

Corollary 14 *Let x and s be examples. If x violates C and s violates C, then $x \cap s$ violates C.*

Proof: Apply Lemma 14 after noting that if s violates C then it also covers C. □

Lemma 15 *If x does not cover C, then for any example s, $x \cap s$ does not violate C.*

Proof: If $antecedent(C) \not\subseteq true(x)$ then $antecedent(C) \not\subseteq true(x) \cap true(s) = true(x \cap s)$. Thus $x \cap s$ does not violate C. □

Lemma 16 *If $x \cap s$ violates C, then at least one of x and s violates C.*

Proof: Observe that $x \cap s$ violates C, and thus $consequent(C) \in false(x \cap s) = false(x) \cup false(s)$. Therefore, $consequent(C)$ is an element of at least one of $false(x)$ and $false(s)$. But since $x \cap s$ violates C, we also have $antecedent(C) \subseteq true(x)$ and $antecedent(C) \subseteq true(s)$. Thus at least one of x and s must violate C. □

2.3 The Algorithm

The ideas behind our algorithm may be understood by considering the problems that arise when we attempt to employ more straightforward approaches. After our algorithm is motivated in this manner, an example run is given. The correctness of the algorithm is demonstrated in the next section; in Section 2.5 we present a more efficient version of the same algorithm.

Let H_* be the target Horn sentence with respect to which equivalence and membership queries are answered. The algorithm is based on the following ideas. Every negative example x violates some clause C of H_*. From x we would like to add the clause C to our current hypothesis, but we cannot exactly determine C from x alone. We know however that $antecedent(C) \subseteq true(x)$, and $consequent(C) \in false(x)$. Thus one approach would be to add to our current hypothesis H all elements of the set

$$clauses(x) = \{(\bigwedge_{v \in true(x)} v) \Rightarrow z : z \in false(x)\}$$

whenever a new negative counterexample x is obtained. Each clause in this set is a possible explanation as to why x is a negative example, and each clause guarantees that in the future x will be classified as a negative example. Moreover, at least one of these clauses is subsumed (i.e.,

logically implied) by some clause of H_*. However, in addition to adding a clause with the "correct" consequent, we may be adding several clauses with the "wrong" consequent. Fortunately, this problem is not of concern because any such clause that is not logically implied by the target formula H_* will eventually be discovered when a *positive* counterexample is produced that does not satisfy the clause. At this point, at least one extraneous clause will be eliminated.[2]

Unfortunately, a simple scenario shows that this straightforward approach is inadequate because we can force it to add exponentially many insufficiently strong clauses. Suppose H_* is defined over the variable set $V = \{a, b_1, b_2, \ldots, b_n\}$ and in fact H_* is just a single clause, C, which is

$$a \Rightarrow \mathbf{F}.$$

Now any example in which a is set to **T** is a negative example. In particular, the example with a set to **T** and the variables $\{b_i\}_{i \text{ even}}$ set to **T** is a negative example. Among the clauses generated from this example is

$$(a \wedge b_2 \wedge b_4 \wedge \cdots \wedge b_n) \Rightarrow \mathbf{F}$$

Now we see a negative example which is identical to the previous negative example except that b_1 is **T** instead of b_2. This example does not violate the first clause that we generated so we are obligated to exclude this negative example by generating clauses for it. Among these clauses is

$$(a \wedge b_1 \wedge b_4 \wedge b_6 \wedge \cdots \wedge b_n) \Rightarrow \mathbf{F}$$

which, like its predecessor, is logically entailed by H_*. The difficulty is now clear: there are exponentially many negative examples with a set to **T** and exactly half of the variables $\{b_i\}$ set to **T** — we are forced to exclude each one with its own clause. We have observed above that by including $clauses(x)$ in the current hypothesis H upon seeing a negative counterexample x, some "good" clause is added to H, namely, the clause $(\bigwedge_{v \in true(x)} v) \Rightarrow consequent(C)$, where C is some clause of H_* that x violates. However, the scenario above demonstrates that the problem with the straightforward approach is that the antecedents of the clauses of generated

[2]Step 8 of the learning algorithm of figure 2.1 incorporates this idea of discarding extraneous clauses when a positive counterexample is produced.

by *clauses*(x) are too long, so this "good" clause is less restrictive than C because its antecedent is more restrictive. Thus, the negative examples that fail to satisfy the (good) added clause may be only a small fraction of those that fail to satisfy C, and the clause added to H is only an "approximation" of C. Consequently, very many such approximations to the target clause C are generated by the examples.

To counter this problem, a second approach might be to find smaller antecedents by using membership queries to set more of the variables to **F** in the negative counterexamples that we are given by any equivalence query. Thus, given a negative counterexample x we set some variable which is currently **T** in x to **F** and ask whether the result satisfies H_*. If it does not, then the result still violates some clause of H_* so we leave the variable set to **F**, otherwise we set the variable back to **T**. We repeat this process until we can set no more variables to **F**. This process can be done quickly and we are left with a minimal negative example. However, a second scenario will disclose the problem with this approach.

Suppose H_* is defined over variable set $V = \{a, b, c, d\}$ and in fact H_* is

$$(b \wedge c \Rightarrow d) \wedge (b \Rightarrow a)$$

Further suppose that we minimize negative examples by trying to set the variables to **F** in alphabetical order. Let **TTTF** (that is, the variables a, b, and c set to **T** and the variable d set to **F**) be the first negative counterexample. We minimize this to **FTFF** and add *clauses*(**FTFF**) to our hypothesis, so H becomes

$$(b \Rightarrow a) \wedge (b \Rightarrow c) \wedge (b \Rightarrow d) \wedge (b \Rightarrow \mathbf{F})$$

Next we see the positive counterexample **TTFF** which simply reduces H to

$$(b \Rightarrow a)$$

Thus we have indeed found a clause of H_*. Next we see the negative counterexample **TTTF**. But this is the same example that we saw at the beginning, and so our algorithm will never terminate; it is forced to find the same clause repeatedly. The difficulty lies in the fact that even though we were given an example that violated the first clause of H_*, our minimization produced

an example that violated the second clause of H_*; and this fact led to non-termination. (One might object that it is foolhardy to decide *a priori* the order in which we will try to set the variables of our negative examples to **F**; rather we should dynamically decide the order in which to minimize a new negative counterexample. However, it appears to be a difficult problem to design a polynomial-time algorithm that is guaranteed to minimize a negative example and simultaneously avoid rediscovering any of the previously found "good" clauses even if is known which clauses in the current hypothesis are genuinely "good".)

As demonstrated by the first scenario, we must reduce the number of variables set to **T** in the negative counterexamples we are given, but as seen in the second scenario we cannot do this in the obvious way. A data-driven approach solves the dilemma. A new negative example is used to attempt to "refine" previously obtained negative examples by intersection (bitwise conjunction) – each such intersection, if it actually contains fewer true variables than the previously obtained negative example, is then tested to see whether it is negative (using a membership query.) If so, it is a candidate to refine the previously obtained negative example.

The algorithm maintains a sequence S of negative examples. Each new negative counterexample is used to either refine one element of S, or is added to the end of S. In order to learn all of the clauses of H_*, we would like the clauses induced by the (negative) examples in S to approximate *distinct* clauses of H_*. This will happen if the examples in S violate distinct clauses of H_*. Overzealous refinement may result in several examples in S violating the same clause of H_*. To avoid this, whenever a new negative counterexample could be used to refine several examples in the sequence S, only the leftmost among these is refined.

By collecting these ideas, the learning algorithm (Figure 2.1) can be described intuitively in the following way. The sequence S of negative examples that the algorithm maintains is used to generate new hypotheses. Each negative example in the sequence can be explained by $O(n)$ different Horn clauses, and each of these possible explanations is placed in the hypothesis. Any clause in the hypothesis which is not logically entailed by the target will be exposed eventually by a positive counterexample. When this positive counterexample appears, the algorithm removes any clause that this example violates from the hypothesis. On the other hand, the hypothesis may also contain some clauses which are insufficiently strong. These clauses will be exposed eventually by a negative counterexample. When this negative counterexample occurs, the algorithm refines the first element of S it can (using an intersection and a membership query)

or it appends the new negative example to the end of S. In either case, after modifying S the algorithm generates a new hypothesis from S. In Section 2.4 it is proved that this process produces in polynomial time a hypothesis which is logically equivalent to the target.

A simulated run of the learning algorithm will now be given. Suppose H_* is defined over the variable set $V = \{a, b, c, d\}$ and in fact H_* is

$$H_* : (a \wedge c \Rightarrow d) \wedge (a \wedge b \Rightarrow c)$$

Initially we set S to be the empty sequence and H to be the null hypothesis

$$S : []$$

$$H : \emptyset$$

Suppose the first counterexample to our equivalence query for H is the negative example **TTTF**. There are no elements of S that we can attempt to refine with this negative example, so we simply append it to the end of the sequence, and since S changed we generate a new hypothesis H by conjoining all of the *clauses*(s) for all $s \in S$. Thus,

$$S : [\mathbf{TTTF}]$$

$$H : (a \wedge b \wedge c \Rightarrow d) \wedge (a \wedge b \wedge c \Rightarrow \mathbf{F})$$

Now suppose the next counterexample to our equivalence query for H is the positive example **TTTT**. This eliminates an extraneous clause from H but does not change S, so we do not generate a new H from S. Thus we now have

$$S : [\mathbf{TTTF}]$$

$$H : (a \wedge b \wedge c \Rightarrow d)$$

For clarity, we will assume for the remainder of this simulated run that we are able to discard immediately any extraneous clauses by positive counterexamples returned from equivalence queries for H, and so we will only show the effect of receiving a (necessarily) negative coun-

HORN

1 Set S to be the empty sequence /* s_i denotes the i-th element of S. */
2 Set H to be the empty hypothesis
3 UNTIL **equivalent**(H) returns "yes" DO
4 /* main loop */
5 Let x be the counterexample returned by the equivalence query
6 IF x violates at least one clause of H
7 THEN /* x is a positive example */
8 remove from H every clause that x violates
9 ELSE /* x is a negative example */
10 BEGIN
11 FOR each s_i in S such that $true(s_i \cap x)$ is properly contained in $true(s_i)$
12 BEGIN
13 query **member**($s_i \cap x$)
14 END
15 IF any of these queries is answered "no"
16 THEN let i be the least number such that **member**($s_i \cap x$) was answered "no"
17 *refine* s_i by replacing s_i with $s_i \cap x$
18 ELSE add x as the last element in the sequence S
19 ENDIF
20 Set H to be $\bigwedge_{s \in S} clauses(s)$, where $clauses(s) = \{(\bigwedge_{v \in true(s)} v) \Rightarrow z : z \in false(s)\}$
21 END
22 ENDIF
23 END /* main loop */
24 Return H

Figure 2.1: Algorithm for Learning Horn Sentences.

terexample. Suppose the next negative counterexample is **TTFT**. We can intersect this with the (first) element of S and get a negative example with strictly fewer variables set to **T** than the (first) element of S had. We do so, replacing this element of S with the result of the intersection. Then, because S changed, we generate a new hypothesis H from S. (Again assuming that all extraneous clauses were subsequently eliminated) we have

$$S : [\mathbf{TTFF}]$$

$$H : (a \wedge b \Rightarrow c)$$

Suppose our next negative counterexample is **TTTF**. We cannot intersect this with any element of S and get a negative example with strictly *fewer* variables set to **T** that that element of S currently has, so we add our new negative counterexample to the end of the sequence S. This changes S, so we regenerate H by conjoining all of the $clauses(s)$ for all $s \in S$. This eventually leaves us with

$$S : [\mathbf{TTFF}, \mathbf{TTTF}]$$

$$H : (a \wedge b \Rightarrow c) \wedge (a \wedge b \wedge c \Rightarrow d).$$

Now suppose our final negative counterexample is **TFTF**. We cannot refine the first element of S with this negative example, but we can refine the second element of S, so we replace the second element of S by the intersection of that element and our negative counterexample. This in turn mandates that we regenerate H from S, which eventually leaves us with

$$S : [\mathbf{TTFF}, \mathbf{TFTF}]$$

$$H : (a \wedge b \Rightarrow c) \wedge (a \wedge c \Rightarrow d)$$

Our final equivalence query for H tells us that we have learned H_*.

Note that the first and fourth negative counterexamples were the same, namely **TTTF**; thus the current hypothesis held by the algorithm is not necessarily consistent with the examples seen so far.

2.4 Correctness and Running Time

We prove that the algorithm of Figure 2.1 correctly terminates in time $\tilde{O}(m^3n^4)$, making $O(m^2n^2)$ equivalence queries and $O(m^2n)$ membership queries. In the next section, a more efficient algorithm improves these bounds to $\tilde{O}(m^2n^2)$,$O(mn)$, and $O(m^2n)$, respectively.

First observe that the algorithm terminates only if the hypothesis and the target Horn sentence H_* are equivalent. Therefore, if the algorithm terminates, it is correct. To show termination in polynomial time we first prove a couple of technical lemmas.

Lemma 17 *For each execution of the main loop of line 3, the following holds. Suppose that in step 5 of the algorithm a negative example x is obtained such that for some clause C of H_* and for some $s_i \in S$, x violates C and s_i covers C. Then there is some $j \leq i$ such that in step 17 the algorithm will refine s_j by replacing s_j with $s_j \cap x$.*

Proof: The proof is by induction on the number of iterations k of the main loop of line 3. If $k = 1$, then the lemma is vacuously true, since the sequence S is empty upon execution of step 5. Assume inductively that the lemma holds for iterations $1, 2, \ldots, k-1$ of the main loop, and assume that during the k-th execution of the loop, at step 5 a negative example x is obtained such that for some clause C of H_* and for some $s_i \in S$, x violates C and s_i covers C. Clearly, if in step 17 of the k-th iteration, the algorithm refines some s_j where $j < i$, then we are done. Suppose that this does not happen. Now by Lemma 13, we know that $s_i \cap x$ is a negative example. It only remains to be shown that $true(s_i \cap x)$ is properly contained in $true(s_i)$, for then s_i will be refined in step 17. Observe that each time the sequence S is modified, step 20 of the algorithm discards the old hypothesis and constructs a new hypothesis H from the elements currently in S. Further observe that during each execution of the main loop of line 3, either S is modified (lines 9-21), or else a clause is removed from H (line 8). Let $j < k$ be the last execution of the main loop of line 3 during which S was modified. Then, during the j-th iteration, line 20 was executed and H was reconstructed from S. At this time a clause $\widehat{C} = \left(\bigwedge_{v \in true(s_i)} v\right) \Rightarrow consequent(C)$ was included in H, where C is the clause that x and s_i both violate. Now C logically implies $\widehat{C}$, so $\widehat{C}$ could not have been removed in line 8 during iterations $j+1, \ldots, k$ of the main loop. Since the equivalence query returns only examples in the symmetric difference between the hypothesis H and the target H_*, a negative example obtained in line 5 satisfies every clause of H. By assumption, x violates C, thus $consequent(C) \in false(x)$.

But now if $true(s_i) \subseteq true(x)$, then x would violate $\widehat{C}$, a contradiction. Therefore, $true(s_i \cap x)$ is properly contained in $true(s_i)$. Thus the algorithm will replace s_i by $s_i \cap x$ in line 17. □

Lemma 18 *Let S be a sequence of elements constructed for the target H_* by the algorithm. Then*

1. $\forall k \forall (i < k) \forall (C \in H_*)$ *if s_k violates C then s_i does not cover C*

2. $\forall k \forall (i \neq k) \forall (C \in H_*)$ *if s_k violates C, then s_i does not violate C.*

Proof: The proof is by induction. We will show that properties 1 and 2 are preserved under any modifications the algorithm makes to the sequence S.

Initially the sequence is empty, so both properties hold vacuously. Now suppose that the properties hold for some sequence, and suppose that the algorithm modifies the sequence in response to seeing the negative example x.

If the algorithm appends x to the sequence as, say, s_t, then suppose by way of contradiction that property 1 fails to hold. Inductively, the only way that property 1 could now fail to hold is if there is some $i < t$ such that s_i covers some clause C of H_* that s_t violates. But this means $s_i \cap x$ violates C. This together with Lemma 17 contradicts the fact that the algorithm did not replace s_j by $s_j \cap x$ for some $j \leq i$. Thus property 1 is preserved.

Now suppose by way of contradiction that property 2 fails to hold. Inductively, the only way property 2 could now fail to hold is if there is some $i < t$ such that s_i and s_t both violate some clause C of H_*. Since s_i violates C it also covers C. Then, by Lemma 17, some s_j with $j \leq i$ would have been refined instead of $x = s_t$ being added to S, a contradiction. Thus property 2 is preserved.

Now suppose that instead of appending x to the sequence, the algorithm replaces some s_k with $s_k \cap x$. Suppose by way of contradiction that property 1 fails to hold. There are two possibilities, either there is some $i < k$ such that s_i covers and $s_k \cap x$ violates some particular clause C of H_* or there is some $i > k$ such that $s_k \cap x$ covers and s_i violates some particular clause C of H_*. If the former case holds, then by Lemma 16 either x violates C or s_k violates C. If x violates C then (since s_i covers C) by Lemma 17 there must be some $j \leq i < k$ such that s_j was refined instead of s_k, a contradiction. On the other hand, if s_k violates C, then the fact that s_i and s_k both violate C contradicts the inductive assumption that property 2 held before the modification.

Now consider the latter possibility, namely that there is some $i > k$ such that $s_k \cap x$ covers and s_i violates some clause C of H_*. Then by (the contrapositive of) Lemma 15, s_k covers C. Since s_i violates C and $i > k$, this contradicts the inductive assumption that property 1 held before the modification. Thus, property 1 is preserved.

Finally, suppose that the algorithm replaces some s_k with $s_k \cap x$ and suppose by way of contradiction that property 2 no longer holds. If this is the case, then there is some $i \neq k$ such that s_i and $s_k \cap x$ both violate some particular clause C of H_*. By Lemma 15, s_k covers C. Further, by Lemma 16, at least one of s_k and x must violate C. If s_k violates C, then the inductive assumption that property 2 held before the modification is contradicted by the fact that s_i also violates C. On the other hand, suppose x violates C. If $i > k$, then the fact the s_k covers C contradicts the inductive assumption that property 1 held before the modification. If $i < k$, then Lemma 17 and the fact that s_i violates (and hence covers) C contradicts the fact that the algorithm did not replace s_j by $s_j \cap x$ for some $j \leq i$. Thus, property 2 is preserved. □

Corollary 19 *At no time do two distinct elements in S violate the same clause of H_*.*

Proof: This is property 2 of Lemma 18. □

Lemma 20 *Every element of S violates at least one clause of H_*.*

Proof: Each of the elements in S is a negative example, thus by Proposition 1, each of the elements violates some clause of H_*. □

Lemma 21 *If H_* has m clauses, then at no time are there more than m elements in the sequence S.*

Proof: This follows immediately from the fact that each of the elements in S violates some clause of H_* but no two elements violate the same clause of H_*. □

Finally, we have our theorem.

Theorem 22 *A Horn sentence consisting of m clauses over n variables can be exactly learned in time $\tilde{O}(m^3n^4)$ using $O(m^2n^2)$ equivalence queries and $O(m^2n)$ membership queries.*

Proof: The only changes to the sequence S during any run of the algorithm involve either appending a new element to S, or refining an existing element. Thus $|S|$ cannot decrease

during any execution of the main loop of the algorithm. But Lemma 21 shows that there are at most m elements of S at any time. Thus line 18 is executed at most m times. Now observe that whenever any element s_i of the sequence S is refined (line 17), the resulting new i-th element is $s_i \cap x$, which, by line 11, must contain strictly fewer variables assigned the value "true" than s_i. This can happen at most n times for each element of S. Thus line 17 is executed at most nm times. Whenever the ELSE clause at line 9 is executed, either line 17 or 18 must be executed. It follows that lines 9-21 are executed at most $nm + m = (n+1)m$ times. Note that this bounds the total number of membership queries made by $(n+1)m^2$.

Next observe that for any element s of S, the cardinality of $\mathit{false}(s)$ is at most $n+1$ (recalling that $\mathbf{F} \in \mathit{false}(s)$). Thus the cardinality of $\mathit{clauses}(s)$ is at most $n+1$. Therefore, the number of clauses in any hypothesis H constructed in line 15 is at most $(n+1)m$. Now, since each positive counterexample obtained in line 5 necessarily causes at least one clause to be removed from H by line 8, the equivalence query can produce at most $(n+1)m$ positive counterexamples between modifications to S. Therefore, line 8 is executed at most $(n+1)^2m^2$ times. Since each execution of line 3 that does not result in termination causes execution of line 8 or lines 9-21, the total number of executions of line 3 (and hence the total number of equivalence queries made) is at most $(n+1)^2m^2 + (n+1)m + 1$.

To complete the proof we need only show that the time needed for each execution of the main loop is $\tilde{O}(n^2m)$. Using the facts (above) that at any time during the execution of the algorithm $|S| \leq m$ and $|H| \leq (n+1)m$, and that each element of H consists of at most $n+1$ variables (antecedent + consequent), it is easily verified that the time needed to execute either of steps 8 and 20 is $\tilde{O}(n^2m)$, and that these steps dominate the time to execute one iteration of the main loop. □

2.5 Improvements to the Algorithm

We describe a more efficient version of the learning algorithm for Horn sentences. There is a natural shorthand notation for propositional Horn sentences obtained by gathering up all the clauses with the same antecedent set and conjoining the consequents. The conjunction of several clauses $C_1, \ldots, C_k$ with the same antecedent will be represented as a *meta-clause* whose antecedent is the common antecedent of the clauses and whose consequent is the conjunction

of *consequent*(C_i) for $i = 1, \ldots, k$. For example, the meta-clause

$$(a \wedge b \wedge d \Rightarrow c \wedge e)$$

is logically equivalent to, and will be used to represent, the conjunction

$$(a \wedge b \wedge d \Rightarrow c) \wedge (a \wedge b \wedge d \Rightarrow e).$$

The new version of the algorithm maintains the current hypothesis as a sequence of meta-clauses, one meta-clause corresponding to each negative example in the sequence S in the previous version. We assume that this representation is used both by the algorithm and for the equivalence queries. (If the equivalence queries require that the representation be strictly a conjunction of Horn clauses, further (straightforward) optimizations must be made to achieve the time bounds below.)

In addition to this shorthand representation, we make use of the observation that once a positive counterexample eliminates a clause, it eliminates any clause with a refined antecedent. For example, the positive counterexample **TTTF** eliminates the clause $(a \wedge b \wedge c \Rightarrow d)$, and also refinements like $(a \wedge b \Rightarrow d)$ and $(b \wedge c \Rightarrow d)$. Thus, when we refine the antecedent of a meta-clause, we do not need to re-introduce possible consequents that have been eliminated.

The effects of counterexamples on meta-clauses can be exemplified as follows, starting with the meta-clause

$$(a \wedge b \wedge c \Rightarrow d \wedge e \wedge f).$$

A positive counterexample causes items to be struck from the consequent, for example, a positive counterexample **TTTTFT** would result in the meta-clause

$$(a \wedge b \wedge c \Rightarrow d \wedge f).$$

A subsequent negative counterexample that refines the corresponding negative example moves variable(s) from the antecedent to the consequent. For example, the negative example **TTFFFF** then results in

$$(a \wedge b \Rightarrow c \wedge d \wedge f).$$

Using suitable data structures, this means that the total processing time spent modifying the hypothesis is $\tilde{O}(mn)$, since each variable can appear in the antecedent, be moved to the consequent, and be deleted, from each meta-clause. A more formal treatment follows.

We use the partial ordering $\leq$ on assignments defined by $x \leq y$ if and only $x_i \leq y_i$ for $i = 1, \ldots, n$. If C is a meta-clause, let $negex(C)$ denote the example that assigns **T** to all the variables in the antecedent of C and **F** to all the other variables. Then $negex(C)$ is the minimum example in the ordering by $\leq$ that violates C. In the new version of the algorithm, H is the conjunction of a sequence of meta-clauses C_i such that $negex(C_i) = s_i$.

We define three meta-clause operations: generating a new meta-clause from a negative example ($new(x)$), reducing a meta-clause with a positive counterexample ($reduce(C, x)$), and strengthening a meta-clause with a negative example ($refine(C, x)$.)

Given a negative example x, define $new(x)$ to be the meta-clause whose antecedent is precisely the set of variables assigned **T** by x and whose consequent is **F**. Note that $negex(new(x)) = x$. For example,

$$new(\mathbf{TFTFT}) = (a \wedge c \wedge e \Rightarrow \mathbf{F}).$$

This operation is used to construct an initial meta-clause from a new negative example. It replaces the operation $clauses(x)$. The consequent **F** is introduced first because the Horn clause with antecedent set A and consequent **F** logically implies every other Horn clause with antecedent set A. The other possible consequents are only introduced if and when the consequent **F** is eliminated by some positive counterexample.

Given a meta-clause C and an example $x > negex(C)$ for which C is false (intuitively, x is a positive counterexample) we define $reduce(C, x)$ to be a meta-clause with the same antecedent as C and consequent defined as follows.

1. If the consequent of C is the logical constant **F**, then the consequent of $reduce(C, x)$ is the conjunction of those variables v_i such that v_i is assigned **F** in $negex(C)$ and v_i is assigned **T** in x.

2. If the consequent of C is not the logical constant **F**, then the consequent of $reduce(C, x)$ is the conjunction of those variables in the consequent of C that are assigned 1 in x.

For example,

$$reduce((a \wedge b \Rightarrow \mathbf{F}), \mathbf{TTTFT}) = (a \wedge b \Rightarrow c \wedge e).$$

Also,

$$reduce((a \wedge b \Rightarrow c \wedge d), \mathbf{TTTFT}) = (a \wedge b \Rightarrow c).$$

Given a meta-clause C and an example $x < negex(C)$ (intuitively, x is a negative counterexample) we define $refine(C, x)$ to be a meta-clause whose antecedent is all the variables assigned **T** by x, and whose consequent is defined as follows.

1. If the consequent of C is the logical constant **F**, then the consequent of $refine(C, x)$ is the logical constant **F**.

2. If the consequent of C is not the logical constant **F**, then the consequent of $refine(C, x)$ is the conjunction of all the variables in the consequent of C and all the variables assigned **F** by x and **T**by $negex(C)$.

For example,

$$refine((a \wedge b \wedge c \Rightarrow \mathbf{F}), \mathbf{TFTFF}) = (a \wedge c \Rightarrow \mathbf{F}).$$

Also,

$$refine((a \wedge b \Rightarrow c \wedge e), \mathbf{TFFFF}) = (a \Rightarrow b \wedge c \wedge e).$$

Note that the possible consequent d is not re-introduced here, having been (presumably) eliminated by a previous positive example.

The new version of the algorithm, shown in Figure 2.2, has the same line numbers as the previous version, for ease of comparison. As for the previous version, if the algorithm halts then its output is correct, so we need only give bounds on its running time.

Since throughout the new version $negex(C_i)$ is equal to s_i in the old version, the same argument shows that if m is the number of clauses of the target H_*, then there are at most m meta-clauses in H at any time. Note that no meta-clause is ever deleted from H.

Consider the career of a particular meta-clause C_i. C_i is initially created in response to a negative counterexample. When C_i is first created, its antecedent consists of a conjunction of variables, and its consequent is the logical constant **F**.

The antecedent of C_i then only changes in response to negative counterexamples, and the change must be to delete one or more variables from the antecedent. Thus, there can be at most n such changes. Since every negative counterexample must either cause the creation of a new meta-clause or refine an existing one, there can be at most $m(n+1)$ negative counterexamples.

```
NewHORN
 1   /* H is a conjunction of meta-clauses C_i */
 2   Set H to be the empty hypothesis
 3   UNTIL equivalent(H) returns "yes" DO
 4     BEGIN /* main loop */
 5     Let x be the counterexample returned by the equivalence query
 6     IF x violates at least one meta-clause of H
 7       THEN /* x is a positive example */
 8         replace C_i by reduce(C_i, x) for every C_i that x violates
 9       ELSE /* x is a negative example */
10         BEGIN
11         FOR each C_i in H such that (negex(C_i)∩x) < negex(C_i)
12           BEGIN
13           query member(negex(C_i)∩x)
14           END
15         IF any of these queries is answered "no"
16           THEN let i be the least number such that member(negex(C_i)∩x) is
               answered "no"
17             replace C_i by refine(C_i, negex(C_i)∩x)
18           ELSE add new(x) as the last meta-clause in H
19         ENDIF
20         /* H is already updated */
21         END
22       ENDIF
23     END /* main loop */
24   Return H
```

Figure 2.2: New Version of Algorithm for Learning Horn Sentences.

The consequent of C_i may change in response to negative or positive counterexamples. The first positive counterexample to C_i changes its consequent from the logical constant **F** to the conjunction of a subset of the variables not in the antecedent of C_i. Negative counterexamples before the first positive counterexample do not change the consequent of C_i - it remains **F**. Subsequent negative counterexamples to C_i may move one or more variables from the antecedent to the consequent of C_i. Positive counterexamples to C_i, after the first, can only remove variables from the consequent of C_i. Every variable can be deleted at most once from the consequent of C_i, and, if the consequent is not the logical constant **F**, at least one variable must remain in it. Thus, C_i can be changed by at most n positive counterexamples. Since every positive counterexample must change one or more meta-clauses, there can be at most mn positive counterexamples.

Since each negative counterexample can cause at most m membership queries, no more than $m^2(n+1)$ membership queries will be made. With a straightforward representation of meta-clauses and assignments as lists of length n, this algorithm can be implemented to run in time $\tilde{O}(m^2n^2)$.

Our analysis of the improved algorithm establishes the following theorem.

Theorem 23 *A Horn sentence consisting of m clauses over n variables can be exactly learned in time $\tilde{O}(m^2n^2)$ using $O(mn)$ equivalence queries and $O(m^2n)$ membership queries.*

2.6 Compression

It is interesting to note that because the membership and equivalence queries are answerable by a polynomially time bounded teacher, it is possible to apply the Horn learning algorithms to obtain a polynomial time algorithm that given an arbitrary Horn sentence produces an equivalent Horn sentence whose size is at most linearly larger than the smallest equivalent Horn sentence.

Corollary 24 *There exists a polynomial time algorithm that given any Horn sentence H over n variables produces an equivalent Horn sentence H' whose size is $O(n)$ times the size of the smallest equivalent Horn sentence.*

Proof: The learning algorithms presented here find a representation with the fewest distinct antecedents, because each negative example violates some clause of the target, there are no

more negative examples in S than there are clauses in the target, and a single antecedent is constructed from each negative example in S. For each antecedent, every consequent supported by the target is added; there are only $O(n)$ possible consequents for a given antecedent. Thus, the size Horn sentence constructed by the learning algorithms is at most $O(n)$ times the size of the smallest representation.

Now, since membership and equivalence queries are answerable in polynomial time, given H as input, we can construct a polynomial time algorithm to answer queries about H posed by the learning algorithms instead of requiring a teacher. The Horn sentence H' produced by the learning algorithm against this substitute teacher is sufficiently small. □

2.7 Hardness Results

If membership queries are not available, it is an open problem whether Horn sentences are PAC learnable or polynomial-time predictable (from random examples alone.) By the reductions of Kearns, Li, Pitt, and Valiant [65] PAC learnability of Horn sentences would imply PAC learnability of general CNF and DNF sentences, and similarly for polynomial predictability.

It is also an open problem whether general CNF or DNF formulas are learnable or polynomially predictable when equivalence and membership queries are available. For any k, let k-quasi-Horn be the class of CNF formulas where each clause contains at most k unnegated literals. Thus 1-quasi-Horn is just the class of Horn sentences, and is learnable using equivalence and membership queries. Here, using a clever observation by Vijay Raghavan about Pitt and Warmuth's requirements for prediction-preserving reductions [86], we show that if an algorithm exists to learn the class of 2-quasi-Horn formulas using equivalence and membership queries then the general class of CNF formulas (and DNF formulas) would be learnable by an algorithm that uses membership and equivalence queries.

Corollary 25 *If the class of 2-quasi-Horn formulas is learnable using membership and equivalence queries, then the class of CNF formulas is learnable using membership and equivalence queries.*

Proof: Let the class of CNF formulas be defined over the variable set $\{x_1, \ldots, x_n\}$. Introduce new variables $\{y_1, \ldots, y_n\}$ such that the negation of y_i will behave like x_i. Let C_* be the

target CNF formula, and everywhere the variable x_i appears replace it with the negation of y_i. Notice that this produces a CNF formula (over the variables $\{x_i\}_{i=1}^n \cup \{y_i\}_{i=1}^n$) that has no unnegated literals, and is thus Horn. Obtain a CNF formula C'_* by conjoining to this formula the n clause pairs $(x_i \vee y_i) \wedge (\neg x_i \vee \neg y_i)$, which require that exactly one of x_i and y_i is set to T in any example that satisfies C'_*. Note that C'_* is a a 2-quasi-Horn formula whose size is only polynomially larger than C_*.

Now consider extending any example e of C_* (over n variables) to an example e' of C'_* (over $2n$ variables) by setting the value of y_i to be the negation of the setting of x_i. The example x satisfies C_* if and only if x' satisfies C'_*. The objective here is that because such a 2-quasi-Horn C'_* exists for *any* C_*, we can employ a 2-quasi-Horn learning algorithm $\mathcal{A}$ to identify C_* by conditioning all examples as above and pretending that the extant C'_* is the actual target. Clearly, this conditioning can be accomplished without knowing C_*.

Now, when $\mathcal{A}$ poses a membership query, if every x_i is set oppositely of its corresponding y_i, then make a membership query about C_* using the settings of the x_i and return that answer to $\mathcal{A}$. Clearly, the answer to these two membership queries must be the same. On the other hand, if some x_i and y_i are set identically, answer the query "no" – this setting violates one of the clauses of the hypothetical C'_* that demands that the x_i and y_i be set oppositely. Thus, answering membership queries correctly for C'_* does not require knowledge of C_*.

When $\mathcal{A}$ poses an equivalence query on the hypothesis H', we know that, if correct, the variables x_i and y_i must behave as negations of one another in the hypothetical C'_*. Thus, obtain H by replacing every variable y_i by the literal $\neg x_i$ and make an equivalence query against C_* about H. If the equivalence query is answered "yes" we are done because H is a CNF over $\{x_i\}_{i=1}^n$ that is logically equivalent to C_*. If the equivalence query is answered "no" and an example e (over n variables) is returned, then e satisfies C_* if and only if e falsifies H. Extend e to an example e' (over $2n$ variables) as described above. Because H was obtained from H' by replacing y_i with $\neg x_i$, e' satisfies H' if and only if e satisfies H, so that e satisfies C_* if and only if e' falsifies H'. But e satisfies C_* if and only if e' satisfies the hypothetical C'_*, so e' satisfies C'_* if and only if e' falsifies H'. That is to say, e' is a counterexample to H' for the hypothetical target C'_*. Therefore, supplying $\mathcal{A}$ with the counterexample e properly answers the equivalence query on H' "no". Thus, answering equivalence queries correctly for C'_* does not require knowledge of C_*.

It now follows that the above construction produces a learning algorithm for the class of CNF formulas from membership and equivalence queries, provided that $\mathcal{A}$ exists. □

The following PAC related corollary, whose proof borrows ideas from the previous corollary, also holds.

Corollary 26 *If the class of 2-quasi-Horn formulas is PAC learnable using membership queries, then the class of CNF formulas is PAC learnable using membership queries.*

Proof Sketch: As in the previous corollary, a 2-quasi-Horn target (over twice as many variables) exists for a given CNF formula. The simulation now is less complicated than before. The randomly generated examples received are transformed into examples of the 2-quasi-Horn as before, the membership queries posed by the 2-quasi-Horn algorithm are answered as before, and there are no equivalence queries to answer. □

Finally, based on the Angluin and Kharitonov result [10], we immediately have the following corollary.

Corollary 27 *Assuming the existence of one-way functions, if the class of 2-quasi-Horn formulas is PAC learnable using membership queries, then the class of CNF formulas is PAC learnable from random examples alone (without membership queries).*

Proof: The Angluin and Kharitonov result says that under the assumption of the existence of one-way functions, membership queries do not help when trying to predict CNF in the PAC setting. □

2.8 Discussion

An exact learning algorithm for propositional Horn sentences using membership and equivalence queries was presented. By the results of Angluin [6, 4] neither type of query alone is sufficient to allow exact learning in polynomial time. The algorithm may be used to obtain an algorithm for PAC learning or polynomial prediction [59, 86] of Horn sentences from randomly generated examples, provided that membership queries are also available to the algorithm.

An interesting open problem is whether the algorithm here can be extended so as to handle restricted types of universally quantified Horn sentences (see the papers of Valiant [99] and

Haussler [57] for related classes of formulas). Although this answer appears to be "no", this class is of significant interest due to its similarity to the language Prolog, and its use in logic programming and expert system design. The pessimism arises from the dramatic change in semantics between propositional and first-order logics. Specifically, the obvious first-order analog of a propositional example is a potentially infinite table specifying truth assignments to every object in the Herbrand universe of the target formula. Infinite sized examples render "polynomial time" meaningless. Even so, in chapters 6 and 7, after discussing alternative types of examples, we will discuss polynomial time learning algorithms for two first-order classes.

Chapter 3

Propositional Horn Sentences and Membership by Logical Entailment

To motivate the distinction between teachers providing standard variable assignment examples and teachers providing formulas labeled according to logical entailment, consider the following two learning tasks involving elephants. The unknown concept of an elephant might be expressed as a boolean formula of a particular form defined over variables which denote real-world attributes. For example, the formula *large* $\wedge$ *grey* $\wedge$ *has-trunk* $\wedge$ *eats-peanuts* might represent exactly the set of all elephants, to the extent that an assignment to the above boolean variables represents an elephant if and only if it satisfies the conjunction.

A related, but perhaps philosophically different problem, is obtained by viewing the target as the description of the only world the learner has at his disposal to explore. Here, an unknown formula ϕ_* may be viewed as a theory about the world, and positive examples α will be statements which follow from the theory. For example, if the theory contained the implications that "large mobile things can crush you," "elephants are large things," and "elephants are mobile," then a positive example would be the entailed sentence "elephants can crush you," a fact that one might wish to know if living near elephants. We seek to construct a theory about the world containing assertions such as "elephants are mobile" and "elephants are large things" given entailed assertions such as "elephants can crush you". Such a problem is central to a number of areas, including expert system design (construct a knowledge base from

example real-world facts) and logic programming (synthesize a Prolog program from example input/output behavior).

Philosophically, the distinction between these two learning tasks can be summarized as follows. Learning from a standard teacher can be thought of as learning to classify. In this setting, the unknown formula is a description of the world and the settings of the variables are worlds which either do or do not satisfy the description; thus the learner is seeing a collection of variable settings and is being asked to learn to classify those settings as "worlds" and "non-worlds". In contrast, learning from an entailed example teacher can be thought of as having just one world whose description is to be deduced from truths about the world.

Though there have been numerous[1] interesting results in learning from standard teachers, there has been relatively little work that investigates learning from entailed example teachers.[2] We present a polynomial time algorithm for learning propositional Horn sentences using membership and equivalence queries from an entailed example teacher using Horn clauses as examples. Our interest in the entailed example teacher was sparked by work in the area of *approximate entailment* [37, 92, 63, 52]. We suggest that our algorithm might be useful in this setting, and leave open the general question of how techniques found in learning algorithms might be helpful in this area.

The current algorithm is in fact an application of the algorithm presented in the previous chapter, but with a number of twists. We will present two different learning algorithms – one of which learns the class directly and one of which applies a prediction preserving reduction to the learning algorithm in the previous chapter. Although the main ideas in the direct algorithm and proof are the same as in the previous chapter, some tricks are needed to overcome a few difficulties. The result is, in fact, a learning algorithm for a different learning problem. As with the standard teacher, the entailment teacher algorithm learns propositional Horn sentences in time polynomial in the size of the sentence to be learned.

Motivating our results, Angluin describes an algorithm for learning propositional Horn sentences, from equivalence, and "derivation" queries [5]. A derivation query allows the learning algorithm to propose a clause C, and in response, is told whether C is "subsumed," "not sub-

[1] See proceedings from *Workshop on Computational Learning Theory* [61, 60, 48, 47, 100, 38].

[2] There has been some work in the area of inductive logic programming that considers learning with entailed first-order atoms [82] rather than clauses.

sumed, but entailed," or "not entailed" by the Horn theory to be learned. Thus, our algorithm strengthens Angluin's result by limiting the types of answers given in response to a query ("entailed" or "not-entailed"). Note that queries answered "subsumed", which we do not allow, give information about the syntactic structure, and not just the logical structure, of the theory to be learned. Eliminating subsumption queries has the appeal that no particular target representation needs to be assumed, as well might be the case when examples are provided from nature. Angluin leaves open the question that this chapter closes.

The recent results in the area of concept learning from examples were catalyzed by the introduction of reasonable formal definitions of efficient concept learning [98, 3], and the development of techniques for constructing efficient learning algorithms and proving their correctness. In contrast, there has been relatively little work that applies these new definitions and techniques to the problem of learning from entailment (but see Section 3.2). In this chapter we describe a polynomial-time algorithm that can learn an unknown propositional Horn sentence φ_* by posing certain natural queries to a teacher.

3.1 Introduction

We consider the following learning protocol: Some propositional Horn sentence T_* is chosen and is unknown to the learner, the learning algorithm is permitted membership and equivalence queries, and the learning algorithm sees Horn clauses labeled according to entailment by T_* as examples. We prove the following:

Theorem 28 *Let T_* be any propositional Horn sentence over n variables. The algorithm entailHORN (figure 3.3), using equivalence queries and membership queries (of entailed clauses) answered with respect to the unknown formula T_*, halts in time polynomial in n and the size[3] of T_*, and outputs a Horn sentence H that is logically equivalent to T_*.*

More briefly, we'll say that entailHORN exactly learns the class of Horn sentences from entailment, using equivalence and membership queries. This learning model is, with one important exception, Angluin's standard, well-investigated model of exact learning from equivalence queries and membership queries [3]. The difference is that here the examples queried by the

[3]The size of a Horn sentence is the number of symbols needed to write the Horn sentence.

learner and the counterexamples to the learner's hypotheses are not assignments to the variables which may or may not satisfy T_*, but rather are Horn clauses which may or may not be entailed by T_*. We note that because Horn clause entailment by a Horn sentence is decidable, entailHORN can be converted to a PAC or on-line learning algorithm provided that membership queries remain available.

3.1.1 Approximate Entailment

Let T_* be a known theory. In general, we would like to answer questions of the sort "Is α entailed by T_*?" Depending on T_* and α, the general question can be undecidable, or at least computationally intractable. Consequently, several researchers have grappled recently with the notion of *approximate* entailment. Rather than answering questions of entailment about the actual theory T_* and query α, Dalal and Etherington propose answering questions of entailment about a variety of strengthenings and weakenings of T_* and α ([37]). They note that soundness or completeness or both can be lost under this protocol. Still, certain combinations of their strengthenings and weakenings do preserve either soundness or completeness and permit the question of entailment for queries to be answered efficiently.

Kautz and Selman construct a protocol in which soundness and completeness are preserved even at the expense of categoricity (i.e., some questions of entailment might be answered "I don't know") ([92, 63]). Specifically, they consider the case where T_* is a propositional CNF theory and suggest approximating T_* by choosing as an upper bound the unique strongest Horn sentence entailed by T_* and choosing as a lower bound a weakest Horn sentence which entails T_*. Any Horn clause entailed by the upper bound is also entailed by T_*, and any Horn clause not entailed by the lower bound is not entailed by T_*. Further, since these bounds are Horn sentences, questions of entailment are efficiently answerable for a large class of formulas. They note that the upper bound, while unique, in some cases is exponentially larger than the actual CNF theory T_*. Similarly they note that the lower bound, while never much larger than T_*, is not necessarily unique.

Greiner and Schuurmans also use the idea of upper bounding and lower bounding a known CNF theory T_* with Horn sentences ([52]). They, however, demand that the upper bound be small. Rather than finding the unique Horn least upper bound (which may be large),

they attempt to find, via hill-climbing, a small Horn upper bound that performs optimally at predicting questions about entailment among all Horn upper bounds that are "similar".

Current attempts to construct the Horn least upper bound have a "generate and test" flavor to them – Horn clauses are "generated", and are "tested" by determining (via resolution) whether or not the CNF theory entails the clause. If so, they are included in the Horn least upper bound.

One problem which arises is that even in cases where the Horn least upper bound is small, there seems to be no guarantee that the number of clauses tested is small.[4] Hence, the number of applications of a possibly exponential time resolution procedure might be much larger than the size of the smallest representation of the Horn least upper bound. Ideally, we would like to derive a Horn least upper bound from a CNF theory in time polynomial in the size of the Horn least upper bound. Although we cannot do this, in Section 3.4 we show how algorithm entailHORN can be used to "lazily" construct a Horn least upper bound of a known CNF theory when queries about entailment are received in an on-line setting and periodically we are informed of some error in our entailment prediction. Further, the number of appeals to a potentially exponential-time resolution procedure is bounded by a polynomial in the size of the smallest representation of the Horn least upper bound, and the particular expression for the Horn least upper bound that is found is also at most polynomially larger than the best such expression.

3.2 Related Work

This work builds on the work of the previous chapter that constructs an algorithm for learning Horn sentences from satisfying assignments. The direct algorithm presented below is in fact an application of either algorithm from the previous chapter, but with a number of twists. The standard variable assignment examples of the previous chapter might be viewed as complete conjuncts (or fundamental products) that are labeled as entailing or not entailing the target. Here we do not care about such examples at all; our examples now are Horn clauses that are labeled as being entailed or not being entailed *by* the target. As with the algorithm of the

[4] It is an easy task to construct a small CNF theory that has a small Horn least upper bound but has an exponential number of entailed Horn clauses.

previous chapter, the current algorithm learns propositional Horn sentences in time polynomial in the size of sentence to be learned.

On the other hand, we also present an algorithm for learning from entailment constructed using either algorithm from the previous chapter as a "black box" once a suitable transformation between the entailed example environment and the standard example environment has been constructed. The transformation involves constructing two polynomial time algorithms that use membership queries and equivalence queries to act as oracles for an algorithm of the previous chapter – one to answer the membership queries of the type asked by that algorithm and one to answer the equivalence queries of the type asked by that algorithm. The task accomplished by each oracle is essentially converting between examples of the type available (which are clauses entailed by exactly one of the target and the current hypothesis) and the type used by the algorithms of the previous chapter.

As discussed above, Angluin's work with entailment and subsumption for propositional Horn sentences [5] is closely related to the work presented in this chapter. Other authors have looked at examples entailed by first-order formulas. For example, Page and Frisch look at atoms labeled according to whether they are entailed by a hidden first order formula ([82]). Their work examines more the effect of multiple occurrences of a variable in the hidden theory and less the effect of connectives in the hidden theory. Our work also uses entailment as the means of labeling examples. However, we use entailed examples to learn propositional formulas where the connectives play a significant role within the hidden formula as well as within the examples themselves.

There are some less closely related first-order results from the field of inductive logic programming where the target is a Prolog program. Shapiro describes a system for learning Prolog programs *in the limit* (i.e., time is an unbounded resource) using atoms entailed by the program to be learned ([95]). Also, Džeroski et al. describe an algorithm that learns k-clause, determinate, function-free, first-order Horn clauses with bounded depth variables by transforming the target into a propositional monotone k-term DNF formula ([41]).

3.3 The Algorithm

We now present two algorithms that learn Horn sentences using membership and equivalence queries from an entailed example teacher using Horn clauses as examples.[5]

3.3.1 Learning by Reduction

First, we present an algorithm for exactly learning Horn sentences by entailed example using membership and equivalence queries; the example class is Horn clauses. In order to demonstrate a reduction technique, we present a slightly inefficient interface which is nothing more than polynomial time transformations to and from the examples suitable for the Horn sentence learning algorithm of figure 2.1 or figure 2.2. Here, HORN refers to either algorithm.

The interface is accomplished by constructing algorithms that simulate the oracles for membership and equivalence needed by the HORN algorithm. The constructed oracles are permitted to use entailed example oracles for membership and equivalence, but otherwise must run in polynomial time.

The constructed oracle simulator for the membership queries asked by HORN will be designated $\in_{\text{STD}}$, and the constructed oracle simulator for equivalence queries asked by HORN will be designated $\equiv_{\text{STD}}$. The entailed example oracle for membership and the entailed example oracle for equivalence used by $\in_{\text{STD}}$ and $\equiv_{\text{STD}}$ are designated $\in_{\text{ENT}}$ and $\equiv_{\text{ENT}}$, respectively. The $\in_{\text{STD}}$ oracle simulator is given in Figure 3.1 and the $\equiv_{\text{STD}}$ oracle simulator is given in Figure 3.2.

We now argue that the answers and examples returned to HORN by these oracles are correct. We note by convention that "$\exists v \notin A$" permits v to be the logical constant **F**. We also note that we are tacitly assuming that we know the names of all the variables used in the hidden concept. By assuming that the variable names are not allowed to change once our learning begins, the oracles constructed can simply restart the HORN algorithm with an updated set of variable names whenever the $\in_{\text{ENT}}$ or $\equiv_{\text{ENT}}$ oracles mention a new variable name. Clearly, the HORN algorithm will need to be restarted at most once for each variable name in the hidden concept.

[5] The teacher could be using Horn sentences as examples – a Horn sentence is entailed if and only if every Horn clause is entailed. Using membership queries about clauses (which can be viewed as degenerate sentences), a Horn clause not entailed by the target can be efficiently extracted from a Horn sentence not entailed by the target.

```
∈STD(x)
1   Let A be the variables set to true in x
2   if ∃v ∉ A such that ∈ENT(A → v)
3      then return "no"
4      else return "yes"
```

Figure 3.1: Membership oracle for HORN.

```
≡STD(H)
1   if ≡ENT(H) = "yes" then return "yes"
2   Let C be the clause returned by ≡ENT(H)
3   Let A be the set of variables in the antecedent of C
4   if C is a positive example
5      then
6         until A stops changing
7            if ∃v ∉ A such that H ⊨ A → v then A = A ∪ {v}
8         Let x be the example formed by setting all the variables in A to T and all
          variables not in A to F
9         return x as a negative example
10     else
11        until A stops changing
12           if ∃v ∉ A such that ∈ENT(A → v) then A = A ∪ {v}
13        Let x be the example formed by setting all the variables in A to T and all
          variables not in A to F
14        return x as a positive example
```

Figure 3.2: Equivalence oracle for HORN.

For purposes of exposition, we routinely treat a complete monomial and the unique variable assignment which satisfies it interchangeably. Likewise, we treat a set of variables and the conjunction of those variables interchangeably.

Lemma 29 *The oracle* $\in_{\mathrm{STD}}$ *correctly answers, in time polynomial in the number of variable names in the hidden concept, any membership query asked by HORN.*

Proof: Clearly $\in_{\mathrm{STD}}$ runs in polynomial time. We must now show that the answer it provides is correct.

The example queried by HORN is a negative example if and only if it falsifies some clause of the hidden concept. An example falsifies a clause, c, of the hidden concept if and only if the example sets every variable in the antecedent of c to **T** and sets the consequent of c to **F**. But in this case, the clause whose antecedent is the set of variables set to **T** by the example and whose consequent is the consequent of c (which cannot be among the variables set **T** by the example) is subsumed[6] by c and is therefore entailed by the hidden target. Necessarily, among the $\in_{\mathrm{ENT}}$ queries asked by $\in_{\mathrm{STD}}$, is this particular clause. □

Lemma 30 *The oracle* $\equiv_{\mathrm{STD}}$ *correctly answers, in time polynomial in the number of variable names in the hidden concept and in time polynomial in the length of the presented concept, any equivalence query asked by HORN.*

Proof: Clearly, the first line of $\equiv_{\mathrm{STD}}$ correctly determines whether the presented concept and the hidden concept are logically equivalent. However, this is not enough. We must also show that any example returned to HORN by $\equiv_{\mathrm{STD}}$ is an example on which the presented concept and the hidden concept disagree.

If the clause C returned by the $\equiv_{\mathrm{ENT}}$ query is a positive example, then it is entailed by the hidden concept and it is not entailed by the presented concept. Thus, step 7 cannot cause the consequent of C to be added to A. However, A does contain the antecedent of C. Thus the example x returned to HORN falsifies C and therefore falsifies the hidden concept. At the same time the construction in step 7 guarantees that x satisfies the presented concept. This because

[6] The meaning of "is subsumed by" used here comes from automated theorem proving; the specific meaning here in this propositional setting is "is formed from a set of literals that is a subset of the literals of". Chapter 7, consistent with the terminology of description logics, uses the phrase differently.

step 7 finds the unique set of the variables that H demands be satisfied when the variables in the antecedent of C are satisfied; if H is falsified when exactly this constructed set of variables is satisfied, then $H \models C$, contradicting the fact that C was a positive counterexample for H. Also note that step 7 runs in time polynomial in the size of presented concept because for any Horn sentence H and any Horn clause A, $H \models A$ can be answered in time polynomial in the sizes of H and A. At most $O(n^2)$ such entailment questions are asked in this step, where n is the number of variable names in the hidden concept.

Similarly, if the clause C returned by the $\equiv_{\mathrm{ENT}}$ query is a negative example, then it is entailed by the presented concept and is not entailed by the target concept. Step 11 constructs an example which satisfies the hidden concept and falsifies the presented concept. The only difference between this step and step 6 is that we cannot answer the series of entailment question ourselves, so we use the $\in_{\mathrm{ENT}}$ oracle to obtain the answers to the required entailment queries.

□

3.3.2 Learning Directly

Next we present an algorithm that does not rely on the existence of an algorithm that learns from a teacher who provides the standard variable assignment examples. Throughout, α (and any variation of α) is a (possibly empty) conjunction of propositional variables. We will frequently treat α as a set; in the case that α is empty, its conjunction represents the logical constant **T**. Throughout the paper, β (and any variation of β) represents a set of at most one propositional variable; in the case the β is empty, it is understood to represent the logical constant **F**. We represent propositional Horn clauses with the schema $\alpha \rightarrow \beta$; we refer to α as the *antecedent* of the clause and to β as the *consequent*.

We use the boldface $\in$ to denote a membership query and the non-boldface $\in$ for normal set membership. As is standard, we use $\models$ to denote logical entailment and use $\cap$ and $\subset$ for set intersection and *proper* subset, respectively. The algorithm entailHORN is shown in figure 3.3.

Two crucial properties of entailHORN are (1) that entailHORN keeps a monotonically growing sequence of antecedents `approxAnts` that provide clues about distinct clauses of the target and (2) that entailHORN provides increasingly better approximations as to the identity of these clauses. The current hypothesis `hypoth` is constructed from the antecedents comprising `approxAnts` by asking for each of these antecedents which consequents are supported by the

target (line 8). Upon receiving an equivalence query counterexample C (line 3), entailHORN computes the (unique) minimal model of **hypoth** satisfying the antecedent of C (line 4) and tries to use this minimal model to chip away excess variables in some antecedent in **approxAnts** (line 6). If this attempt fails, entailHORN appends this minimal model to **approxAnts** as the initial approximation to the antecedent of some clause of the target for which no information (in the form of a counterexample) has yet been provided (line 7).

Before arguing that entailHORN is correct and runs efficiently, we need the following supporting definitions and lemmas.

Definition 31 *If ϕ is a satisfiable formula, ψ is a falsifiable formula, and $\phi \models \psi$, then ϕ is said to entail ψ* non-trivially.

Next we present a series of lemmas which will enable us to prove the correctness of entailHORN.

Lemma 32 *If $(\alpha \rightarrow \beta) \models (\alpha' \rightarrow \beta')$ non-trivially, then $\alpha \subseteq \alpha'$ and $\beta \subseteq \beta'$.*

Proof: If $\alpha \not\subseteq \alpha'$, then set all variables in α' to **T** and all other variables to **F**. Since there is some variable in $\alpha - \alpha'$ that is set **F**, this setting satisfies $\alpha \rightarrow \beta$. Since $\alpha' \rightarrow \beta'$ is falsifiable, this assignment falsifies it. But then $(\alpha \rightarrow \beta) \not\models (\alpha' \rightarrow \beta')$, a contradiction.

Similarly, if $\beta \not\subseteq \beta'$, then assign **F** to those variables in β' and **T** to all other variables. Because $\alpha' \rightarrow \beta'$ is falsifiable, this assignment falsifies it. However, since $\beta \not\subseteq \beta'$, there is some variable in β that is assigned **T** so that $\alpha \rightarrow \beta$ is satisfied, a contradiction. □

Lemma 33 *Let T be a Horn sentence. If $T \models (\alpha \rightarrow \beta)$ non-trivially, then α is a superset of the antecedent of some clause of T.*

Proof: If T contains some clause with an empty antecedent, then the lemma holds vacuously.

For the case when T contains no clauses with empty antecedent, suppose by way of contradiction that the lemma does not hold. Assign **T** to all variables in α and **F** to all other variables. (Since $\alpha \rightarrow \beta$ is falsifiable, this assignment falsifies it.) Now notice that since by supposition no antecedent in T is contained in α, at least one variable in each antecedent of T is **F**, thus every antecedent in T is **F**, and so T must be satisfied. Therefore $T \not\models \alpha \rightarrow \beta$, a contradiction. □

Naming the hidden target Horn theory T_*, we will now argue correctness.

BEST COPY AVAILABLE

entailHorn

1 **approxAnts** = $\emptyset$

2 **hypoth** = $\emptyset$

3 while **hypoth** is not equivalent to the target, let C be the counterexample returned by the equivalence oracle

4 $\quad \tilde{\alpha} = \{\beta : \textbf{hypoth} \models (\text{ant}(C) \rightarrow \beta)\}$

5 $\quad$ if $\exists \alpha \in$ **approxAnts** such that

- $\alpha \cap \tilde{\alpha} \subset \alpha$
- $\text{RHS}(\alpha \cap \tilde{\alpha}) \neq \emptyset$

6 $\quad\quad$ then replace the first such α in **approxAnts** by $\alpha \cap \tilde{\alpha}$

7 $\quad\quad$ else append $\tilde{\alpha}$ to **approxAnts**

8 $\quad$ **hypoth** = $\{\alpha \rightarrow \beta : \alpha \in \text{approxAnts}, \beta \in \text{RHS}(\alpha)\}$

9 return **hypoth**

Figure 3.3: The algorithm entailHORN used to learn Horn sentences.

RHS(α)

1 return $\{v : v \notin \alpha, ^a \in (\alpha \rightarrow v)\}$

[a] It is permissible for v be the logical constant **F**.

Figure 3.4: Find all consequents of entailed clauses having a given antecedent.

Lemma 34 *At all times during the run of entailHORN,* T_* $\models$ hypoth.

Proof: The subroutine RHS is used to explicitly check that each clause of hypoth is entailed by T_*. A clause $\alpha \rightarrow \beta$ is placed in hypoth only if $\beta \in \text{RHS}(\alpha)$ (line 8). This can only occur if $\in(\alpha \rightarrow \beta)$, hence $T* \models (\alpha \rightarrow \beta)$. Since T_* entails every clause of hypoth, $T_* \models$ hypoth. □

Corollary 35 *Any counterexample to* hypoth *with respect to* T_* *must be entailed by* T_* *but not by* hypoth.

Notice that we can easily determine whether the target is unsatisfiable by first making an equivalence query on an unsatisfiable Horn sentence, so for expository purposes we henceforth assume that the target is satisfiable. Also, if the target is a tautology then the first equivalence query made by entailHORN will be correct, so we henceforth also assume that the target is falsifiable. Finally, under these assumptions, since the counterexamples obtained by equivalence queries (line 3) are entailed by exactly one of two satisfiable Horn sentences, counterexample clauses are neither unfalsifiable nor unsatisfiable.

Lemma 36 *If* $\alpha \rightarrow \beta$ *is any counterexample of* hypoth *with respect to* T_*, *then there is some clause* $\alpha_* \rightarrow \beta_*$ *of* T_* *such that* $\alpha_* \subseteq \alpha$ *and* $\beta_* \notin \alpha$.

Proof: By Corollary 35, $T_* \models (\alpha \rightarrow \beta)$ and hypoth $\not\models (\alpha \rightarrow \beta)$. Because hypoth $\not\models (\alpha \rightarrow \beta)$, $\alpha \rightarrow \beta$ must be falsifiable. Consider the variable assignment that makes α **T** and makes all other variables **F**; this assignment falsifies $\alpha \rightarrow \beta$. Since $T_* \models \alpha \rightarrow \beta$ this assignment must falsify T_*. But T_* is falsified if and only if some clause $\alpha_* \rightarrow \beta_*$ of T_* is falsified. But this assignment falsifies $\alpha_* \rightarrow \beta_*$ if and only if $\alpha_* \subseteq \alpha$ and $\beta_* \notin \alpha$. □

The following lemma shows that no matter what counterexample entailHORN receives from the equivalence oracle, the antecedent that entailHORN actually uses to modify approxAnts is the antecedent of a (possibly different) counterexample.

Lemma 37 *If* $\alpha \rightarrow \beta$ *is a counterexample returned to entailHORN by the equivalence query oracle in line 3, then the* $\tilde{\alpha}$ *computed in line 4 is such that* $\tilde{\alpha} \rightarrow \beta$ *is also a counterexample.*

Proof: By Corollary 35, $T_* \models (\alpha \rightarrow \beta)$ and hypoth $\not\models (\alpha \rightarrow \beta)$. Clearly $\tilde{\alpha}$ contains α, thus $T_* \models (\tilde{\alpha} \rightarrow \beta)$. We now show that if hypoth $\models (\tilde{\alpha} \rightarrow \beta)$, then hypoth $\models (\alpha \rightarrow \beta)$.

Suppose that **hypoth** $\models (\tilde{\alpha} \rightarrow \beta)$. Let e be any variable assignment that satisfies **hypoth**. If e does not satisfy α, then e satisfies $\alpha \rightarrow \beta$. On the other hand, if e does satisfy α then, since e satisfies **hypoth**, by construction e also satisfies $\tilde{\alpha}$. Since **hypoth** $\models (\tilde{\alpha} \rightarrow \beta)$ and since e satisfies **hypoth** and $\tilde{\alpha}$, e must satisfy β. But then e satisfies $\alpha \rightarrow \beta$. Thus **hypoth** $\models (\alpha \rightarrow \beta)$.

But **hypoth** $\not\models (\alpha \rightarrow \beta)$. Thus **hypoth** $\not\models (\tilde{\alpha} \rightarrow \beta)$. Therefore, $\tilde{\alpha} \rightarrow \beta$ is a counterexample to **hypoth** with respect to T_*. □

We want to show that **hypoth** is becoming increasingly closer to T_*; to do this we have the following definition which will help describe how far **hypoth** is from T_*.

Definition 38 *For* $\alpha \in$ **approxAnts**, *we say that* α properly covers *a clause* $\alpha_* \rightarrow \beta_*$ *if* $\alpha_* \subseteq \alpha$ *and* $\beta_* \notin \alpha$

We are now prepared to state and prove our main lemma.

Lemma 39 *If* α_1 *and* α_2 *are antecedents in distinct positions of the sequence* **approxAnts**, *then* α_1 *and* α_2 *each properly cover some clause of* T_*, *but* α_1 *and* α_2 *do not properly cover the same clause of* T_*.

Proof: First observe that for every α in **approxAnts** there is some β such that $T_* \models (\alpha \rightarrow \beta)$. This is because line 6 replaces an element of **approxAnts** only if the replacement, $\alpha \cap \tilde{\alpha}$, is such that $T_* \models (\alpha \cap \tilde{\alpha}) \rightarrow \beta$ for some β. On the other hand, line 7 simply appends the antecedent of the (possibly implicit (Lemma 37)) counterexample constructed in line 3. But by Corollary 35, this counterexample is entailed by T_*. Since every α, when placed in **approxAnts**, was a counterexample to **hypoth**, every α is the antecedent of some clause that is non-trivially entailed by T_*. Thus, by Lemma 33, every α in **approxAnts** properly covers some clause of T_*. Hence α_1 properly covers some clause of T_* and α_2 properly covers some clause of T_*.

We now show that the clauses properly covered by α_1 and α_2 are different. Suppose not and consider the first iteration of the main loop in which this property fails to hold. Let α_1 and α_2 be two elements of **approxAnts** that properly cover the same clause $\alpha_* \rightarrow \beta_*$ of T_*. Without loss of generality, assume that this iteration of the main loop caused α_2 to appear in **approxAnts**. There are two ways in which this iteration could place α_2 in **approxAnts** – either α_2 was appended to **approxAnts** or α_2 was the result of intersecting $\tilde{\alpha}$ with some α_2' that was in **approxAnts** in the previous iteration of the loop.

In the former case, since α_1 and α_2 (which is actually $\tilde{\alpha}$) both properly cover $\alpha_* \rightarrow \beta_*$, $\alpha_1 \cap \tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$, thus $\beta_* \in \mathrm{RHS}(\alpha_1 \cap \tilde{\alpha})$ and so $\mathrm{RHS}(\alpha_1 \cap \tilde{\alpha}) \neq \emptyset$. Thus, for entailHORN not to have used $\tilde{\alpha}$ to replace α_1 in **approxAnts**, it must be that $\alpha_1 \cap \tilde{\alpha} = \alpha_1$. But, since α_1 properly covers $\alpha_* \rightarrow \beta_*$, $\beta_* \in \mathrm{RHS}(\alpha_1)$, and since α_1 was in **approxAnts** when **hypoth** was constructed for this iteration **hypoth** contains $\alpha_1 \rightarrow \beta_*$. Thus $\beta_* \in \tilde{\alpha}$, contradicting the assumption that $\tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$. Thus, $\tilde{\alpha}$ could not have been appended to **approxAnts**.

In the latter case, where $\tilde{\alpha}$ was used to refine some antecedent already in **approxAnts**, we have that $\alpha_2 = \tilde{\alpha} \cap \alpha_2'$ where α_2' is some element of **approxAnts** in the preceding iteration of the loop. Either α_1 precedes α_2 in **approxAnts**, or α_1 follows α_2 in **approxAnts**.

Consider the case in which α_1 precedes α_2 in **approxAnts**. If $\alpha_2' \cap \tilde{\alpha}$ properly covers α_*, then either α_2' properly covers $\alpha_* \rightarrow \beta_*$ or $\tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$. But if α_2' properly covered $\alpha_* \rightarrow \beta_*$, then the inductive hypothesis is violated because α_1 and α_2' properly covered the same clause in the previous iteration. Thus, it must be the case that $\tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$. Since entailHORN did not use $\tilde{\alpha}$ to refine α_1, it must be the case that $\tilde{\alpha} \cap \alpha_1 = \alpha_1$. But, in this case $\alpha_1 \subseteq \tilde{\alpha}$ and since α_1 properly covers $\alpha_* \rightarrow \beta_*$, then $\beta_* \in \mathrm{RHS}(\alpha_1)$, and since α_1 was in S when **hypoth** was formed for this iteration, $\beta_* \in \tilde{\alpha}$, contradicting the assumption that $\tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$.

Last consider the case where α_1 follows α_2 in **approxAnts**, in which case entailHORN did not try to refine α_1 with $\tilde{\alpha}$. If $\alpha_2' \cap \tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$ then either α_2' properly covered $\alpha_* \rightarrow \beta_*$ or $\tilde{\alpha}$ properly covered $\alpha_* \rightarrow \beta_*$. If α_2' properly covered $\alpha_* \rightarrow \beta_*$, then the inductive hypothesis is violated because in the previous iteration α_1 and α_2' properly covered the same clause. Thus it must be the case that $\tilde{\alpha}$ properly covers $\alpha_* \rightarrow \beta_*$, that $\alpha_* \subseteq \alpha_2'$, and that $\beta_* \in \alpha_2'$; otherwise, $\alpha_2' \cap \tilde{\alpha}$ could not properly cover $\alpha_* \rightarrow \beta_*$ and yet have α_2' not properly cover $\alpha_* \rightarrow \beta_*$. Now by assumption α_1 properly covers $\alpha_* \rightarrow \beta_*$. Consider all antecedents used by entailHORN to create or refine (by intersection) the position in **approxAnts** currently occupied by α_1. Since $\alpha_* \subseteq \alpha_1$, every antecedent used by entailHORN in the position currently occupied by α_1 must have contained α_*. Further, since α_1 does not contain β_*, one of these antecedents must not have contained β_*. Call the first such antecedent $\tilde{\alpha}'$. Now observe that since α_2 contains both α_* and β_*, we know that every antecedent that has occupied the position currently occupied by α_2' must have contained α_* and β_*. But then for every such predecessor,

α_2'', of α_2' we have that $\alpha_2'' \cap \tilde{\alpha}'$ is a proper subset of α_2'' because α_2'' contains β_* but $\tilde{\alpha}'$ does not. Further, because RHS($\alpha_2'' \cap \tilde{\alpha}'$) properly covers $\alpha_* \rightarrow \beta$ we have that $\beta_* \in$ RHS($\alpha_2' \cap \tilde{\alpha}'$), and therefore RHS($\alpha_2' \cap \tilde{\alpha}'$) is non-empty. This contradicts the assumption that $\tilde{\alpha}'$ was used to refine the position currently occupied by α_1.

Thus the antecedents in **approxAnts** properly cover distinct clauses of T_*. □

We now state our main theorem.

Theorem 40 *Let V be a finite set of propositional variables. The class of Horn formulas using only variables from V is polynomial time learnable from entailed examples using equivalence and membership queries.*

Proof: It is easily verified that each step in the **while** loop of entailHORN takes time at most polynomial in $|V|$ and size of the unknown Horn formula. We need only argue that there are only polynomially many iterations of the loop.

By Lemma 39, every antecedent in **approxAnts** properly covers some antecedent in T_* and no two antecedents in **approxAnts** properly cover the same antecedent of T_*. Thus **approxAnts** never contains more elements than there are clauses in T_*. Furthermore, entailHORN never deletes any element of **approxAnts**, at worst it replaces some element of **approxAnts** by a proper subset of that element. This can happen at most as many times as there are variables in V. □

We immediately have the following corollaries.

Corollary 41 *Let V be a finite set of propositional variables. The class of Horn formulas using only variables from V is polynomial time learnable using examples and membership queries using Horn sentences as examples.*

Proof: (Sketch) At least one Horn clause of a counterexample Horn sentence must be suitable as counterexample for entailHORN. □

Corollary 42 *Let V be a finite set of propositional variables. The class of Horn formulas using only variables from V is polynomial time PAC learnable using random examples and membership queries using Horn clauses as examples.*

Corollary 43 *Let V be a finite set of propositional variables. The class of Horn formulas using only variables from V is polynomial time PAC learnable using random examples and membership queries using Horn sentences as examples.*

3.3.3 Membership Queries are Necessary

In contrast to Corollary 41, we show that equivalence queries do not wield sufficient power alone to allow exact learning of Horn sentences from entailment using Horn sentences as examples. We also give evidence that PAC learning is similarly crippled without membership queries. This shows that the membership queries used in the previous section are necessary.

Before presenting the result, we define a *monotone negative CNF formula* as a conjunction of clauses containing only negated literals. To show the necessity of using membership queries, we start with the following lemma. The theorem following the lemma is the result we seek in this section.

Lemma 44 *There exists an exact learning algorithm for Horn sentences that uses equivalence queries to an entailed example teacher using Horn sentences as examples, only if there exists an exact learning algorithm for monotone negative CNF formula that uses equivalence queries to a standard teacher.*

Proof: Any assignment to a set of propositional variables can be encoded as a conjunction of literals over the variables – if the variable x_i is to be set **T** then place x_i in the conjunction, otherwise place $\neg x_i$ in the conjunction. Clearly only the desired assignment satisfies this conjunction. Now observe that every monotone negative CNF formula is a Horn sentence, and assume that $\mathcal{A}$ is an algorithm that learns Horn sentences from entailment using only equivalence queries. Let C_* be the target monotone negative CNF formula.

When $\mathcal{A}$ makes an equivalence query for some hypothesis H', form H by deleting any positive literals[7] in H' and make an equivalence query against C_*. If the query is answered "yes" then we are done because H is logically equivalent to C_*. If a positive counterexample e_p is returned (which by definition falsifies H, and therefore satisfies all the negated variables of one of its clauses), then the monotonicity of C_* guarantees that changing any variable setting in e_p from **T** to **F** results in a positive example of C_*. So, if e_p satisfies H', e_p must make **T**

[7] If this produces a clause with no literals, then take H to be an unsatisfiable formula.

any unnegated variables appearing in any clause in which the remaining (negated) variables are made *true*; obtain e'_p by setting any such variable to **F**. Then e'_p falsifies H' and H so that like e_p, e'_p is also (positive) counterexample to H against C_*. On the other hand, if a negative counterexample e_n is returned (which by definition satisfies H), since H was obtained by simply deleting literals from (already satisfied) clauses of H', e_n satisfies H'.

It now follows, that by returning the counterexample (e_n, e_p, or e'_p, as appropriate) encoded as a conjunction of literals satisfied only by the counterexample produces a learning algorithm for the class of negative monotone CNF formulas. □

Angluin [6] has shown that monotone DNF formulas (DNF formulas that have only unnegated variables) are not learnable from equivalence queries alone, even when the learner is allowed to make queries about *arbitrary* DNF formulas. Our theorem now follows immediately.

Theorem 45 *Horn sentences cannot be exactly learned (with equivalence queries, but without membership queries) from an entailed example teacher using Horn sentences as examples.*

Proof Sketch: The dual of monotone negative CNF is monotone DNF; simply negating the label of an example provides the correct transformation. □

We next comment on PAC learning (without membership queries) Horn sentences from an entailed example teacher; we have the following lemma and corollary, which relate PAC entailed Horn learnability to standard DNF PAC learnability. The standard PAC learnability of DNF remains the central open problem in computational learning theory.

Corollary 46 *Learning Horn sentences with equivalence queries from an entailed example teacher using Horn sentences as examples is as hard a PAC learning arbitrary DNF formulas from a standard teacher.*

Proof Sketch: Because Kearns et al. [65] have shown that PAC learning DNF without membership queries is no easier than PAC learning monotone DNF without membership queries, the construction reducing the exact monotone DNF standard learning problem to the exact Horn entailment learning problem leading up to Theorem 45 suffices to reduce the PAC (without membership queries) monotone DNF standard learning problem to the PAC Horn entailment learning problem. Thus a PAC Horn entailment learning algorithm would produce a standard PAC DNF learning algorithm. □

3.4 Application to Approximate Entailment

We now consider the application of these entailment results outside the field of computational learning theory by considering the problem of finding the Horn least upper bound of a known CNF theory, where the Horn least upper bound is defined to be the (unique) logically strongest Horn sentence entailed by the CNF theory. Even when the Horn least upper bound for some theory is small, current methods do not guarantee that a small least upper bound will be found without post-processing the set of entailed Horn clauses found with an algorithm such as the propositional Horn compression result of Corollary 24. Such an approach does not lessen the number of resolution proofs needed in the first place to obtain the set of entailed clauses. In contrast the entailment algorithm suggested below will never approximate the actual Horn least upper bound by any Horn sentence having more distinct antecedents than the actual Horn least upper bound. This implies that our algorithm will never approximate the actual Horn least upper bound by a Horn sentence that has more than n times as many clauses – and hence is more than n^2 times as large – as the actual Horn least upper bound where n is the number of variables in the theory. This suggests the approach of using entailHORN to find the Horn least upper bound, using the known CNF theory to answer membership and equivalence queries. However, although the final Horn sentence is guaranteed to be small, this approach too, could result in far too many resolution applications. Instead, we consider a setting in which the CNF theory is manipulated only when necessary (we consider running entailHORN in a "lazy" manner), and the environment is used to provide counterexamples to the correctness of the currently hypothesized Horn least upper bound. We consider two settings in which such an approach seems plausible.

Consider a setting in which an agent who happens to live near elephants is provided with some theory ϕ that describes the world. Our goal is to have the agent maneuver in the world without getting crushed. Among the ways in which our goal can be achieved are (1) have the agent cower in a corner predicting that everything will crush him, (2) in each situation determine from ϕ whether he will be crushed, or (3) tractably approximate ϕ by ϕ' and have him update ϕ' only when he observes (unpredicted by ϕ') something being crushed.

Clearly solution (1) is uninteresting. Solution (2) potentially suffers from the uninteresting features of solution (1) because in the cases where ϕ is intractable, the agent will, in effect,

spend most of his time in a corner predicting that he will be crushed searching for a proof to the contrary. Solution (3) seems plausible in that the agent will spend much of his time exploring the world and only retreat to a corner to spend time manipulating ϕ when nature deems it necessary. We refer to the agent operating under solution (3) as a *lazy agent*. A bit more formally,

Definition 47 *Let ϕ be a (propositional) CNF theory, let $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$ be a set of unlabeled clauses whose entailment by ϕ is to be predicted, and let $\Upsilon = \{v_1, v_2, \ldots\}$ be a set of* urgent *clauses labeled according to entailment by ϕ. Let $\xi_1, \xi_2, \ldots$ be a sequence defined over elements of $\Sigma \cup \Upsilon$. A* lazy *agent is an agent who uses a prediction strategy that*

1. *predicts the entailment of each ξ_i "quickly",*
2. *updates the prediction strategy only upon incorrectly predicting an element of Υ, and*
3. *predicts that ϕ entails ξ_i only if indeed $\phi \models \xi_i$*

Intuitively, the longer lived lazy agent is the one who predicts that more things are entailed. The scare quotes around the word "quickly" in the definition are intended to bias the agent toward using some ϕ' that is a tractable (yet small relative to ϕ) approximation to ϕ, if such a ϕ' exists. Updating ϕ' only upon seeing (labeled) elements of Υ is intended to capture the idea that only those predictions whose outcomes are observable in the world are important enough for the agent to spend time adjusting his prediction strategy to predict them correctly; it can be argued that the agent is concentrating on the facts that are relevant to the environment he is exploring.

Given this definition and discussion, an agent who uses entailHORN to acquire the Horn least upper bound to ϕ and uses intermediate hypotheses `hypoth` to predict the entailment of ξ_i seems a satisfactory lazy agent. Any clause entailed by the Horn least upper bound of ϕ must be entailed by some Horn clause entailed by the Horn least upper bound. It is an easy matter to test all of the (at most n) largest Horn clauses that entail the clause to see which ones are not entailed by `hypoth` and from among those use resolution on ϕ to find some new Horn clause to use as a counterexample for entailHORN to update `hypoth`. Further because entailHORN is an exact learning algorithm, we can treat it as an on-line algorithm with bounded mistakes. This means that on Horn clauses of Υ the agent will make at most a number of predictive

errors polynomial in the size of the Horn least upper bound of ϕ. This is a quantitative means of measuring the amount of time spent on resolution over ϕ.

As a second setting, consider the work by Greiner and Schuurmans, which assumes an agent who receives a sequence of randomly generated clauses ([52]). For each clause, the agent determines (via resolution on ϕ) whether the clause is entailed, and if necessary, adjusts his approximation ϕ'' to the Horn least upper bound ϕ', to correct his prediction. However, the agent constrains ϕ'' to have size at most k, so that prediction remains efficient. To accomplish their goal, they define a set of transformations between Horn sentences of size at most k. In the end, they show that they find a ϕ'' that is locally optimal with respect to the neighborhood defined by the transformations. Hence, they may fail to find a global optimum, even if there is one of size at most k.

In contrast, we consider an agent who is given a set of randomly generated clauses, labeled by entailment according to ϕ. Since our algorithm produces a sequence of Horn sentences of monotonically increasing size, we can easily determine whether there is *any* Horn sentence of size k consistent with the sample. If the true Horn least upper bound has size at most k, or if the best size k Horn least upper bound approximation is consistent with the sample, then our algorithm will find a comparably performing Horn sentence having size at most kn^2. In this setting too, we are able to quantitatively specify the resolution work we must do by noting that resolution over ϕ is required only to answer membership queries, and our algorithm will make at most a number of membership queries that is polynomial in kn^2 and the size of the sample.

3.5 Discussion

Using techniques very similar to Chapter 2, our work closes a question in computational learning theory left open by [2, 5]. The result is an algorithm that by asking natural questions is guaranteed to produce a Horn sentence consistent with a set of clauses labeled as to whether entailed by an unknown Horn sentence; furthermore, the algorithm is guaranteed to take time at most polynomial in the size of the smallest among all such consistent Horn sentences.

This new algorithm is then applied to the problem of approximate entailment found in the field of machine learning. Here, this work adds to the approaches found in the literature. Our approach is to confine the appeals to resolution to only those places where our polynomial time

learning algorithm specifically asks about the entailment of a specific Horn clause. Our goal is to accelerate the construction of a tractable Horn upper bound of a known CNF theory by providing a sharp focus for the resolution machinery. A side effect of our learning algorithm is that it implicitly maintains a monotonically increasing lower bound on the size of the Horn least upper bound of the CNF theory.

We hope techniques from computational learning theory will continue to find fruitful application in the field of machine learning. For example, can efficient learning algorithms further limit the number of appeals to or further limit the focus of resolution in constructing a tractable, useful approximation to a given CNF theory?

Chapter 4

Membership by Subsumption

What formal utility can be imparted to an expert's response *time* when presented with a question? Computer Science is replete with examples of caching for speed improvement. Computer memory hierarchies remember the most recently accessed data item. Operating systems remember the most recently accessed fragment of code. Caching is founded on the principle that what was useful recently will be useful shortly.

We might imagine that a domain expert operates in a similar fashion; faced with a problem, the expert may start with those ideas that proved useful very recently, and in this way he can be seen as caching recent results for use in the near future. Notice that this view assumes that the expert *does not* reason from first principles right away – indeed, at any given moment his description of the world may be quite different than the shortest possible description. However, because the expert uses knowledge that has proven useful recently, a problem that is very similar to a recent problem can be solved quite quickly given the expert's current description of the world, whereas that same new problem might require significant effort to solve given only the smallest possible world description. Thus, the expert's current world description is related to which problems he can solve quickly – *independent of his ability to articulate that description.*

In the previous two chapters, we have considered two notions of example labeling. We have seen variable assignment examples labeled according to whether the target is satisfied, and we have seen clause examples labeled according to entailment by the target. In this chapter we consider a third kind of example labeling that seeks to capture the *efficiency* of an expert in answering questions. Here we imagine that the expert has reached some fixed, cached repre-

sentation of the world and he answers question according to that representation. We explore the interaction of a learner with such an expert. Our goal is to learn a fixed snapshot of the expert's cache at some moment in time, believing that by doing so we are capturing not only a description of the world, but also the proven efficiency of the expert in reasoning about the world.

Angluin [5] provided an algorithm for learning a restricted class of propositional Horn sentence which uses what she terms a *request for hint query*. The answer to this query tells the learner whether his chosen clause is subsumed by some clause of the target. In terms of an expert/apprentice interaction, we view this as the apprentice being told that his question is trivially answered by comparing the clause to the set of clauses in a proven, useful representation – that is, the expert can answer the question quickly (as opposed to taking time to construct a complicated proof).

We present a learning algorithm that uses subsumption queries and equivalence queries. We then show that the algorithm is a polynomial time learning algorithm for any class of CNF formulas that is closed under projection, closed under clause deletion, and has a polynomial time solution to the satisfiability problem. To illustrate the utility of this generic algorithm, we demonstrate that it learns a class of formulas constructed by Boros et al. [27] that properly contains the class of Horn sentences and was not previously known to be learnable.

4.1 Definitions

In order to formally model this expert/apprentice relationship, we need to specify precisely what is to be learned by the apprentice, what information is available to the apprentice, and the types of questions the apprentice is permitted to ask. Here we assume that the expert's description of the world belongs to some class $\mathcal{F}$ of propositional CNF formulas. We assume that the apprentice knows $\mathcal{F}$; however, we assume that the apprentice does not know the expert's world description itself. The apprentice's goal is to learn the expert's target description.

If we expect the apprentice to correct an erroneous hypothesis of the world, we must show him that his hypothesis is, in fact, in error. The means by which we allow the apprentice access to errors in his hypothesis is once again an equivalence query.

To consult with the expert, we allow the apprentice *subsumption queries*. Recall that a CNF formula is a conjunction of disjunctions of literals and that for propositional clauses C and C', C' subsumes a clause C exactly when the set of literals in C' is a subset of the literals in C. A subsumption query occurs when the apprentice chooses a clause and asks the expert whether the clause is subsumed by some clause of the target. The expert answers either "yes" or "no". Thus, subsumption queries can actually be viewed as membership queries using as examples clauses labeled according to *subsumption* by the expert's description of the world. Notice that the expert can answer the subsumption query in time polynomial in the product of the the lengths of the apprentice's clause and the target.

To obtain polynomial time learnability for a class $\mathcal{F}$ of CNF formulas, we depend on $\mathcal{F}$ possessing three properties – the satisfiability problem for $\mathcal{F}$ is decidable in polynomial time, $\mathcal{F}$ is closed under variable projection, and $\mathcal{F}$ is closed under clause deletion. We now define these properties.

Definition 48 *Let $\mathcal{F}$ be a class of formulas, and let $\phi \in \mathcal{F}$. The* satisfiability question *for ϕ is "Does there exist a truth assignment for the variables appearing in ϕ that satisfies ϕ?" The* satisfiability problem for $\mathcal{F}$ is decidable in polynomial time *if there exists an algorithm that correctly answers the satisfiability question for any $\phi \in \mathcal{F}$ and runs in time polynomial in the length of ϕ.*

Definition 49 *Let $\mathcal{F}$ be a class of formulas. We say that $\mathcal{F}$ is* closed under variable projection *if for any $\phi \in \mathcal{F}$ and any variable v appearing in ϕ, fixing the truth value of v to be either* **T** *or* **F** *produces a formula $\phi' \in \mathcal{F}$ such that the size of ϕ' is no greater than the size ϕ.*

Definition 50 *Let $\mathcal{F}$ be a class of CNF formulas. We say the $\mathcal{F}$ is closed under clause deletion if deleting any clause from any $\phi \in \mathcal{F}$ produces a formula $\phi' \in \mathcal{F}$.*

We close this section with a word on notation. We represent clauses C in an implicative form denoted $\alpha \rightarrow \beta$ where α is the conjunction of variables occurring negated in C and β is the disjunction of variables occurring unnegated in C.

4.2 The GENERIC Algorithm

This section proves our main result of this chapter, but first we need a general property relating entailed clauses to subsumption.

Lemma 51 *Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be any CNF formula, and let $\alpha \rightarrow \beta$ be any clause C entailed by ϕ. Then there exists some $C_i = \alpha_i \rightarrow \beta_i$ in ϕ such that C_i subsumes $\alpha \rightarrow \beta$.*

Proof: Let M be the variable assignment that sets every variable in α to **T** and all other variables to **F**; clearly, M falsifies C. Since $\phi \models C$, it must be the case that M also falsifies ϕ, and therefore M must falsify some clause $C_i = \alpha_i \rightarrow \beta_i$ of ϕ. This means that every variable in α_i is set **T** and every variable in β_i is set **F**. But then α must contain every variable in α_i and none of the variables in β_i, so $\alpha_i \rightarrow \beta_i$ is the clause whose existence is asserted by the statement of the lemma. □

Armed with the ability to ask subsumption queries, we now present a learning algorithm for any class $\mathcal{F}$ of formulas that is closed under variable projection, closed under clause deletion, and for which the satisfiability problem is polynomial time decidable. We have the following theorem.

Theorem 52 *Let $\mathcal{F}$ be any class of CNF formulas such that*

- *$\mathcal{F}$ is closed under variable projection,*
- *$\mathcal{F}$ is closed under clause deletion, and*
- *the satisfiability problem for $\mathcal{F}$ is decidable in polynomial time*

Then $\mathcal{F}$ is polynomial time learnable given equivalence and subsumption queries.

Proof: Consider the algorithm GENERIC shown in figure 4.1. Initially h is the universally true hypothesis, thus $\phi \models h$. Every clause conjoined to h in line 17 is subsumed by some clause of ϕ, therefore $\phi \models h$ after every update. Thus, every counterexample obtained in line 2 must be a clause C that h improperly fails to entail. Thus, there must be a satisfying assignment for h that falsifies C. This satisfying assignment is computed in the loop beginning in line 7.

Now, in line 13, after the satisfying assignment has been computed, T and F contain the variables assigned **T** and **F**, respectively. Since this assignment falsifies C and $\phi \models C$, this

assignment falsifies ϕ. Therefore $\phi \models (\bigwedge_{x \in T} x) \rightarrow (\bigvee_{y \in F} y)$. Now by Lemma 51 there is some β' such that $(\bigwedge_{x \in T}) \rightarrow \beta'$ is subsumed by some clause of ϕ; certainly $\bigvee_{y \in F}$ contains that β', and so some clause of ϕ subsumes $(\bigwedge_{x \in T}) \rightarrow (\bigvee_{y \in F})$.

The loop beginning in line 13 removes variables from F one at a time while ensuring that some clause of ϕ still subsumes the resulting clause. Thus this loop finds a minimal set F, i.e., finds *exactly* some β' that is actually the β_i for some clause C_i of ϕ.

Similarly, the loop beginning in line 15 produces a minimal set T such that $(\bigwedge_{x \in T} x) \rightarrow (\bigvee_{y \in F} y)$ is subsumed by some clause of ϕ, i.e., T will contain exactly the variables of α_i for some clause C_i of ϕ whose β_i is exactly contained in F. In other words, the clause added to h in line 17 is some actual clause of ϕ. To see that this is not the same as some clause already in h, notice that the assignment computed in the loop at line 7 satisfies every clause already in h, but this assignment falsifies the clause added to h in line 17.

In regard to running time, since every iteration of the `while` loop in line 2 produces a new clause of the target, there are no more iterations than there are clauses of the target. In the body of the loop, constructing the satisfying assignment for h deserves consideration. Since h contains a subset of the clauses of ϕ, it must be that $h \in \mathcal{F}$ because $\mathcal{F}$ is closed under clause deletion. Similarly, provisionally setting a variable to **T** in line 8 produces a formula in $\mathcal{F}$ no larger than h because $\mathcal{F}$ is closed under variable projection. Last, the test as to whether the variable can be set **T** in line 8 can be done in polynomial time because satisfiability for $\mathcal{F}$ is decidable in polynomial time. For the remainder of the `while` loop body, minimizing the sets T and F requires examining each variable only once. □

4.2.1 A Note on Disposing of Equivalence Queries

As mentioned earlier, it is possible (and perhaps appealing) to replace the equivalence queries with randomly generated examples. If our apprentice simply observes nature and verifies that his hypothesis correctly predicts[1] each observation, then any misprediction can be treated as a counterexample to what would have been an equivalence query. Assuming the expert's description of the world is accurate, the apprentice will make only a number of mistakes polynomial in the size of the expert's description. Thus the apprentice can acquire the expert's efficient

[1] Because $\mathcal{F}$ is closed under variable projection and the satisfiability problem for $\mathcal{F}$ is decidable in polynomial time, prediction of an observation – ie, entailment by the hypothesis – is efficiently decidable for the apprentice.

GENERIC

```
1   Set h to be the empty hypothesis
2   While EQ(h) returns a counterexample α → β
      /* Falsify α → β */
3     Set T to be the set of variables in α
4     Set F to be the set of variables in β
5     Set all variables in α to T and all variables in β to F and let h' be the result of
      projecting these variable setting out of h
6     Let V be the set of unassigned variables in h'
      /* Find a satisfying assignment for h' */
7     For each v ∈ V
8       If h' is satisfiable when v is T
9         Set v T, and project v out of h'
10        Place v in T
        else
11        Set v F, and project v out of h'
12        Place v in F
      /* Remove extraneous variables from the consequent */
13    For each variable v in F
14      If SUBSUME((⋀_{x∈T} x) → (⋁_{y∈F−{v}} y)), remove v from F
      /* Remove extraneous variables from the antecedent */
15    For each variable v in T
16      If SUBSUME((⋀_{x∈T−{v}} x) → (⋁_{y∈F} y)), remove v from T
      /* Add the new target clause to the hypothesis */
17    h := h ⋁((⋀_{x∈T} x) → (⋁_{y∈F} y))
```

Figure 4.1: The GENERIC learning algorithm.

80

description of the world without demanding that the expert answer any equivalence queries, or indeed, without demanding that the expert even be able to articulate his description.

4.3 A Newly Learnable Class

Boros et al. [27] define a class of CNF formulas for which the satisfiability problem is decidable in polynomial time. This class, which we call BCH, is defined next.

Definition 53 *Let V a set of propositional variables. Partition V into two disjoint sets X and Y. The class BCH is the set of arbitrary conjunctions of clauses over V such that*

- *No clause contains more than two unnegated variables,*
- *No clause contains more than two Y variables, and*
- *No clause contains both a Y variable and an unnegated X variable.*

This class contains functions that are representable neither as Horn sentences nor as 2-CNF formulas[2]. For example, taking $X = \{a\}$ and $Y = \{b, c\}$ shows that the formula $(\neg a \vee b \vee c)$ is in BCH, although it is representable neither as a Horn sentence nor as a 2-CNF formula.

Taking $X = V$ shows that BCH contains Horn sentences. The first condition in the definition is satisfied because Horn sentences permit at most one unnegated variable per clause; the last two conditions are trivially satisfied because there are no Y at all.

Taking $Y = V$ shows that BCH contains 2-CNF formulas. The first two conditions of the definition are satisfied because no clause of a 2-CNF formula has more than two variable; the last condition is satisfied because there are no X variables.

Thus, BCH properly contains the union of Horn sentences and 2-CNF formulas. Although this class was not previously known to be learnable, we have the following result.

Corollary 54 *The class BCH is exactly learnable (using time polynomial in the expert's description's size) from subsumption and equivalence queries.*

Proof: Boros et al. showed that the satisfiability problem for BCH is decidable in polynomial time. Next, observe that BCH is closed under variable projection and clause deletion. Thus by Theorem 52, GENERIC is a polynomial time learning algorithm for BCH. □

[2]The class 2-CNF consists of conjunctions of clauses, each clause containing at most two literals.

4.4 Queries About Proof Length

As a final note we consider the notion that an expert may be able to deduce that some fact follows from his description of the world by constructing a simple proof quickly, but beyond some point the complexity of constructing a proof that a particular fact follows from his description exceeds his abilities. We model this provable/unprovable by the length of the proof; we assume that the expert can construct a proof of length k almost instantaneously. In such a situation, the expert's "yes" response actually means that the clause presented by the appretice is subsumed by some clause whose proof is at most k resolutions long. We call this a *proof-length k query.*

By adding the property of being closed under resolution to the hypothesis of Theorem 52 we easily obtain the following corollary.

Corollary 55 *Let k be fixed. Let $\mathcal{F}$ be any class of CNF formulas such that*

- *$\mathcal{F}$ is closed under resolution,*
- *$\mathcal{F}$ is closed under variable projection,*
- *$\mathcal{F}$ is closed under clause deletion, and*
- *the satisfiability problem for $\mathcal{F}$ is decidable in polynomial time*

Then given equivalence and proof-length k queries, $\mathcal{F}$ is exactly learnable in time polynomial in the expert's description's size and exponential in k.

Proof: (Sketch) Since $\mathcal{F}$ is closed under resolution, assume that all resolvents of proofs of length at most k are added to the target and apply Theorem 52 to this padded target. □

4.5 Discussion

This chapter provides a sufficient set of conditions under which a polynomial time learning algorithm using equivalence and subsumption queries is guaranteed to exist. Which, if any, of these conditions is also necessary?

Theorem 52 can be viewed in different ways. Strictly speaking from the view of computational learning theory, the subsumption queries used by GENERIC cheat – they explicitly request information about the *representation* of the target rather than merely information about the

function represented by the target. An *entailment query* – "Is this clause entailed by the target?" – is a non-cheating query related to subsumption. Is there a simple set of sufficient conditions under which a theorem analogous to Theorem 52 holds where entailment rather than subsumption queries are used?

On the other hand, questions of entailment may demand lengthy proofs from the expert, whereas, operationally speaking, answers to questions of subsumption would manifest themselves as the *amount of time* it takes the expert to answer an entailment question. Thus from a practical standpoint, any delay from the expert in answering "Is this clause true?" can be interpreted as an answer of "no" to the question of "Is this clause subsumed?" Thus, by taking advantage of an expert caching an *efficient* description (as opposed to a minimal description) the apprentice acquires an *accurate* and *efficient* description from the expert – even when the expert is unable to articulate that description. Along this line of discussion, an obvious question to investigate is psychological; do experts cache efficient descriptions in representation languages amenable to acquisition by GENERIC?

Chapter 5

Consistently Ignorant Teachers: Membership by Consensus

Most of the theoretical work models the interaction between the learner and the environment by an omniscient oracle (or teacher) that classifies all objects as positive or negative examples of the concept to be learned. Thus, it is assumed that there is a well-defined border separating positive examples from negative ones. In practice, though, classification is often unclear. For example, suppose a robot operating in an assembly plant must determine whether a part is defective. While some parts will be clearly defective and some clearly not defective, there may be some parts for which the teacher cannot decide. As another example, there are situations in which the classification of some objects is not well defined: An algorithm designed to read handwritten cheques will likely encounter many handwritten characters that look somewhat like a "4", and somewhat like a "9". In such cases, where even an expert does not have the knowledge to classify all objects, determining which objects are unclassifiable seems at least as important as determining the classifications of objects which *are* classifiable. From the learner's perspective, the regions of the example space that defy classification create a blurry border between the positive and negative examples that the learner must determine.

In this chapter, we introduce a new formal learning model in which the teacher (or environment) with which the learner interacts has incomplete information about the target function due to intrinsic uncertainty or due to gaps in the teacher's knowledge. The key requirement we place on the teacher is that all examples (or objects) labeled with "?" (indicating unknown

classification) are consistent with the teacher's background knowledge about the class to which the unknown function belongs. In particular, the classification of any example labeled with "?" should not be determinable from the positive and negative examples, and knowledge of the concept class. (Thus the teacher is "consistently ignorant".) Observe that the examples labeled with "?" can be arbitrarily far away from the original border — they are just required to be consistent with the other labels and the background knowledge of the class. The goal of the learner will be to learn a good approximation to the knowledge of the teacher. Namely, the learner must construct a *ternary* function (i.e. with values {0,1,?}) that, with high probability, classifies most randomly drawn examples exactly as the teacher does.

Next we introduce the notion of an *agreement* of concepts from a concept class $\mathcal{C}$, and show that the problem of learning concepts $f \in \mathcal{C}$ from a consistently ignorant teacher can be modeled as the problem of learning agreements of concepts from $\mathcal{C}$. As a third characterization, we show that any blurry concept can be represented as the agreement of a finite union and intersection of concepts from $\mathcal{C}$. We then show that for any concept class $\mathcal{C}$ for which PAC learning algorithms are known, these algorithms can be used to build an algorithm for learning the agreement of nested concepts from $\mathcal{C}$. For the problem of learning the agreement of concepts from $\mathcal{C}$ that are not necessarily nested, we show that if the intersection and union of arbitrarily many concepts from $\mathcal{C}$ is learnable, then $\mathcal{C}$ is learnable from a consistently ignorant teacher. While, often, algorithms are not known for learning unions and intersections of concepts from $\mathcal{C}$, under certain conditions it is still possible to learn the agreements of concepts from $\mathcal{C}$. For example, consider a class $\mathcal{C}$ for which the intersection of concepts from $\mathcal{C}$ is learnable, yet there is no known algorithm to learn the union of concepts from $\mathcal{C}$. In some cases it may still be possible to learn $\mathcal{C}$ from a consistently ignorant teacher by using information gained by learning the intersection of concepts in the agreement to aid in learning the union of these concepts. The learner's ability to use intersection (union) information to obtain positive results for learning unions (intersections) of concepts from classes for which no algorithms are currently known is intriguing. To illustrate the limits of this approach, we show that learning the agreement of an arbitrary number of Horn sentences is as hard as learning DNF. Thus, assuming DNF cannot be learned in the standard model (permitting membership queries), propositional Horn sentences, while learnable in standard models from omniscient teachers, cannot be learned from consistently ignorant teachers.

5.1 Background and Related Work

Most previous research on concept learning assumes for any $f \in \mathcal{C}$ and $x \in \mathcal{X}$ that either $f(x) = 1$ or $f(x) = 0$. In these situations the border between the positive and negative examples is well defined. There has been work addressing the issue of mislabeled training examples [14, 69, 97, 64] and some addressing the issue of noise in the attributes [94, 51, 74]. In these situations, the border between the positive and negative examples may appear blurry to the learner, but this is just the result of the noise process that has been applied to the properly labeled example. There has also been some work considering learning from noisy membership queries [50, 91].

Angluin and Slonim [12] introduced a model of *incomplete membership queries* in which each membership query is answered "don't know" with a given probability. Furthermore, this information is persistent—repeatedly making a query that was answered with "don't know" always results in a "don't know" answer. As in their work, one of our goals is to model the situation in which the teacher responding to the learner's queries is not omniscient. Observe, that in Angluin and Slonim's model since the teacher is *randomly* fallible, there is no guarantee that all of the teacher's knowledge about the target concept is used in answering queries. For example, it is possible that their teacher knows that poodles are mammals, but responds with "don't know" when asked if a french poodle is a mammal.[1] Further, their result for learning monotone DNF depends very heavily on this inconsistency of the teacher. In the context of monotone DNF, our consistency requirement manifests itself as follows: The teacher should know that adding positive attributes to an already positive example yields a positive example. (Dually for negative examples.) Thus, in the standard boolean lattice defined over variable assignments, all positive examples are above all unknown examples, which, in turn, are above all negative examples. In Angluin and Slonim's algorithm for learning monotone DNF, if the teacher replies that $f(x) =?$ then the learner samples below x in the boolean lattice for some (known) positive example y, implying that x is a positive example. If none are found, the learner concludes with high probability that x is a negative example. Thus, the teacher's ignorance is not consistent with the knowledge that the target function is monotone; the learner can

[1] In our view, the notion of an incomplete membership oracle seems to better model noise than it models incomplete knowledge. Indeed, they note that their algorithm for learning monotone DNF with an incomplete membership oracle can be used to learn monotone DNF with random one-sided errors.

determine the underlying boolean function by deducing what the teacher does not (but should) know. In our model, this would not be possible, as the teacher's lack of knowledge is consistent; the best that the learner can do (and what we demand that the learner do) is to learn which examples are positive, which negative, and which are unknown.

In other related work, Kearns and Schapire [67] generalized the PAC setting to non-binary values using Haussler's framework [56]. They define a *p-concept* in which each example $x \in \mathcal{X}$ has some probability $p(x)$ of being classified as positive. An observation consists of an example x drawn randomly according to D and then independently classified as positive with probability $p(x)$ and negative with probability $1 - p(x)$. In their model, the goal of the learner is to make optimal predictions, or more commonly, to accurately predict $p(x)$ for all $x \in \mathcal{X}$. The goal of the learner in our proposed model has similarities with the p-concepts model. However, here we are interested in learning problems for which the learner need just determine whether $p(x) = 0$, $p(x) = 1$, or $0 < p(x) < 1$. (If a written numeral is sometimes identified as "4" and sometimes as "9", the learner just wants to know this—it does not need to determine what percentage of the population would call the numeral each value.) Similarly, our work differs from the literature on fuzzy sets in that we do not quantify degrees of membership. (It differs perhaps even more in our application of the learning models from computational learning theory.)

5.2 The Model of a Consistently Ignorant Teacher

We now formally define our model of learning from a consistently ignorant teacher. A *blurry* ternary concept $f_?$ is created by taking any f from the *base class* $\mathcal{C}$ and changing a set of examples $Q \subseteq \mathcal{X}$ from their current value to "?" indicating that the teacher does not know their classifications. Furthermore, we require that this be done consistent with the knowledge that f was chosen from $\mathcal{C}$: If every concept $f \in \mathcal{C}$ consistent with the labels of examples from $\mathcal{X} - Q$, labels q as positive (respectively, negative), then $f_?$ cannot label q as "?". More formally:

Definition 58 *Let $f_? : \mathcal{X} \to \{0, 1, ?\}$, and let $P = \{x \mid f_?(x) = 1\}$, $N = \{x \mid f_?(x) = 0\}$, and $Q = \{x \mid f_?(x) = ?\}$. Then $f_?$ is a* blurry concept *for $\mathcal{C}$ if for every $q \in Q$, there exists functions f_0 and f_1 in $\mathcal{C}$ such that: (1) for all $x \in P, f_0(x) = f_1(x) = 1$, (2) for all $x \in N, f_0(x) = f_1(x) = 0$, and (3) $f_0(q) = 0 \neq 1 = f_1(q)$. We define the blurry concept class $\mathcal{C}_? = \{f_? \mid f_?$ is a blurry concept for $\mathcal{C}\}$.*

BEST COPY AVAILABLE

Thus for any concept class $\mathcal{C}$, the class $\mathcal{C}_?$ contains exactly those blurry concepts that can be generated from some $f \in \mathcal{C}$. For a target function f, we say that an example $x \in \mathcal{X}$ is a positive example if $f(x) = 1$, is an unknown example if $f(x) = ?$, and is a negative example if $f(x) = 0$. We assume that random examples are chosen (by nature) from an unknown, arbitrary, distribution D, and are then given a label from $\{0, 1, ?\}$ by the teacher, and presented to the learner. Using the obvious extension of the PAC and PAC with membership query models we say that the learner has successfully learned $f_? \in \mathcal{C}_?$ if with probability at least $1 - \delta$, the (ternary) hypothesis output by the learner has probability at most ϵ of disagreeing with $f_?$ on a randomly drawn example from D. If such a polynomial-time learning algorithm exists, we say that the blurry class $\mathcal{C}_?$ is learnable, or equivalently, that the class $\mathcal{C}$ is learnable from a consistently ignorant teacher, in the PAC or PAC with membership query models. Finally, note that one way a hypothesis h might err, is if $f_?(x) =?$ and $h(x) \neq ?$. Thus, "?" does *not* mean "don't care".

5.3 An Alternate Formulation of The Model

To understand some complexity issues involved in learning from consistently ignorant teachers, we consider when $\mathcal{C}$ is the class of pure conjunctive concepts (monomials)—each concept is a simple conjunction of variables or their negations. Let P, Q, and N be the set of positive, unknown, and negative examples, respectively, for some blurry monomial. In this case, it is straightforward to show that P must be representable as a (nonblurry) monomial m. Further, it is not difficult to show that $P \cup Q$ can be represented by a unate[2] DNF that contains only those literals appearing in m (provided P is not empty). These observations are sufficient to construct a PAC with membership query algorithm to learn the class of blurry monomials (for which P is nonempty): run a known algorithm for learning (non-blurry) monomials [98] to learn the set P of positive examples, and at the same time run a known learning algorithm for unate DNF [9] to learn the set $P \cup Q$ of nonnegative examples. Then Q and N can be easily determined from knowledge of P and $P \cup Q$.

Is this a polynomial time algorithm? It depends on our choice of complexity parameters. By definition, there is some "underlying" boolean monomial m (of size $|m| \leq n$, the number

[2] A *unate* formula is one in which no variable appears both negated and unnegated.

of variables) that has been turned into a blurry concept. So perhaps $|m|$ is a reasonable size measure for this blurry concept, and the above algorithm needs to run in time polynomial in $|m|$ to be considered efficient. However, as we observed, the learning problem is not that of determining some underlying boolean concept, but that of determining the ternary blurry concept, which requires learning N, and thus indirectly, $P \cup Q = \mathcal{X} - N$. A particularly nasty choice of "?" examples can result in a unate DNF describing the set $P \cup Q$ that has a number of terms exponential in n (hence, exponential in the size of any monomial from $\mathcal{C}$). Instead of relying then, on the size of some underlying concept, we capture the complexity of a blurry concept by reformulating the notion as that of an *agreement* of base concepts, and let the complexity rely on the complexity of the individual base concepts forming the agreement. Let F be a finite set of boolean functions. The function Agree_F is a ternary function whose classification on example $x \in \mathcal{X}$ is given by

$$\text{Agree}_F(x) = \begin{cases} 1 & \text{if } f(x) = 1 \text{ for each } f \in F, \\ 0 & \text{if } f(x) = 0 \text{ for each } f \in F, \\ ? & \text{otherwise.} \end{cases}$$

We now argue that the problem of learning agreements of concepts from $\mathcal{C}$ is equivalent to learning $\mathcal{C}$ from a consistently ignorant teacher, or equivalently, learning the blurry class $\mathcal{C}_?$. The notion of an agreement of base concepts has independent interest, as it models a type of unanimous vote of independent agents. The following lemma shows that being consistently ignorant is no different than being indecisive in the face of competing hypotheses.

Lemma 57 *For any class $\mathcal{C}$ of boolean concepts, the blurry class $\mathcal{C}_? = \{Agree_F \mid F \subseteq \mathcal{C}\}$.*

Proof: We first show that any $f_? \in \mathcal{C}_?$ can be expressed as an agreement of concepts from $\mathcal{C}$. Let $P = \{x \mid f_?(x) = 1\}$, $Q = \{x \mid f_?(x) = ?\}$, and $N = \{x \mid f_?(x) = 0\}$. We construct a set of concepts F from $f_?$ such that for all $x \in \mathcal{X}$, $f_?(x) = \text{Agree}_F(x)$. Initially, let $F = \emptyset$. Now for each $x \in \mathcal{X}$ for which $f_?(x) = ?$, we add to F the pair of functions f_0 and f_1 described in Definition 56. By definition, f_0 and f_1 exist for each $x \in Q$. Since all concepts placed in F correctly classify all examples in P, it follows that for any x such that $f(x) = 1$, $\text{Agree}_F(x) = 1$. Likewise, for all x such that $f(x) = 0$, $\text{Agree}_F(x) = 0$. Finally, observe that for any x such that

$f(x) =?$ we have placed in F two concepts (f_0 and f_1) that disagree on the classification of x and thus $\text{Agree}_F(x) =?$.

We now show that for any $F \subseteq \mathcal{C}$ there is a blurry concept $f_? \in \mathcal{C}_?$ that is logically equivalent to Agree_F. Select any $f \in F$ as the target function from which $f_?$ will be created. Observe that by the definition of Agree_F, for all examples x that are classified by Agree_F as positive or negative, $\text{Agree}_F(x) = f(x)$. Let $Q \subseteq X$ be the examples classified as "?" by Agree_F. Since for any $x \in Q$ there must be two concepts in F that classify all examples in $\mathcal{X} - Q$ correctly but classify x differently, it follows that for each $q \in Q$, $f(q)$ cannot be computed from $\{(x, f(x) \mid x \in \mathcal{X} - Q\}$. Thus the blurry concept $f_?$ obtained from f by changing the examples in Q to "?" is logically equivalent to Agree_F. □

Hence, the problem of learning blurry concepts $\mathcal{C}_?$ generated from a base class $\mathcal{C}$ is equivalent to the problem of learning the agreements of sets of concepts from the base class $\mathcal{C}$, (and equivalently, learning $\mathcal{C}$ from a consistently ignorant teacher). Using this correspondence, we obtain a complexity measure for the size of $f_?$: First, define the representation size of a subset of concepts F to be $\sum_{f \in F} |f|$. Now define the size of $f_? \in \mathcal{C}_?$ (denoted by $|f_?|$) to be the minimum, over all subsets $F \subseteq \mathcal{C}$ for which $\text{Agree}_F = f_?$, of the representation size of F.

5.4 Positive Results for Learning Agreements

We show that PAC and PAC with membership query learning algorithms can be designed to learn from consistently ignorant teachers. We first consider the problem of learning the agreement of a pair of nested concepts. We show that if both concepts are chosen from classes for which learning algorithms exist, then we can use these algorithms to obtain an algorithm for learning the agreement of the functions. We then present a general result addressing how known algorithms for learning from omniscient teachers can be applied to learn from consistently ignorant teachers even when the base functions are not nested. We apply this technique to obtain positive results for learning (under certain conditions) the agreement of monomials, monotone DNF formulas, and DNF formulas with a constant number of terms.

5.4.1 Learning Agreements of Nested Concepts

Observe that a concept $f \in \mathcal{C}$ can be viewed as a characteristic function denoting the subset of examples from $\mathcal{X}$ that f classifies as positive. Thus for two concepts f_1 and f_2, we write $f_1 \subseteq f_2$ if the set of positive examples of f_1 is a subset of the positive examples f_2. Given a set of concepts $F = \{f_1, \ldots, f_k\}$ we say that these concepts are *nested* if $f_1 \subseteq f_2 \subseteq \cdots \subseteq f_k$. Observe that in such cases $\text{Agree}_{\{f_1,\ldots,f_k\}} = \text{Agree}_{\{f_1,f_k\}}$ and thus, without loss of generality, we consider learning the agreement, $\text{Agree}_{\{f_s,f_g\}}$, of two nested functions f_s and f_g (s and g for "specific" and "general"). Suppose these are chosen, respectively, from known polynomial-time PAC with membership query learnable concept classes $\mathcal{C}_S$ and $\mathcal{C}_G$. Then the learning algorithms for $\mathcal{C}_S$ and $\mathcal{C}_G$ can be used to learn the following blurry concept class:

$$\text{Nested}_?\langle \mathcal{C}_S, \mathcal{C}_G \rangle = \{\text{Agree}_{\{f_s,f_g\}} \mid f_s \in \mathcal{C}_S, f_g \in \mathcal{C}_G, \text{and } f_s \subseteq f_g\}.$$

Theorem 58 *If $\mathcal{C}_S$ and $\mathcal{C}_G$ are PAC with membership query (respectively PAC) learnable concept classes, then* $\text{Nested}_?\langle \mathcal{C}_S, \mathcal{C}_G \rangle$ *is PAC with membership query (respectively PAC) learnable.*

Proof: Let $\mathcal{A}_S$ (respectively $\mathcal{A}_G$) be the PAC with membership query algorithm for learning $\mathcal{C}_S$ (respectively $\mathcal{C}_G$). We obtain an algorithm $\mathcal{A}$ (figure 5.1) to learn $\text{Nested}_?\langle \mathcal{C}_S, \mathcal{C}_G \rangle$ by simultaneously running $\mathcal{A}_S$ treating "?" as "$-$" to obtain h_S, and running $\mathcal{A}_G$ treating "?" as "$+$" to obtain h_G, demanding error at most $\epsilon/2$ with confidence at least $1 - \delta/2$ from each. Then output $h = \text{Agree}_{\{h_S,h_G\}}$ as the final hypothesis.

We now argue that the hypothesis h output by $\mathcal{A}$ has error at most ϵ with probability at least $1 - \delta$. By the correctness of $\mathcal{A}_S$, the probability that the error of h_S is greater than $\epsilon/2$ is less than $\delta/2$. Likewise, the probability that the error of h_G is greater than $\epsilon/2$ is less than $\delta/2$. Clearly the error of h is at most the error of h_S plus the error of h_G, and thus the probability that the error of h is at most ϵ is at least $1 - \delta$. Since $\mathcal{A}_S$ and $\mathcal{A}_G$ run in polynomial time, it immediately follows that $\mathcal{A}$ runs in polynomial time. Finally, note that $\mathcal{A}$ only makes a membership query when either $\mathcal{A}_S$ or $\mathcal{A}_G$ does. □

We note that, under the assumption that the evaluation problems for C_S and C_G are decidable in polynomial time, the stronger exact learning version of Theorem 58 can be obtained – a "?" example returned from an equivalence query are given to $\mathcal{A}_S$ when it would serve as a

Learn-Agreement-Nested-Concepts(F, ϵ, δ)

1 Let $F := \{f_s, f_g\}$ such that $f_s \subseteq f_g$.

2 Let $\mathcal{A}_S$ be the PAC with membership query learning algorithm for $\mathcal{C}_S$.

3 Let $\mathcal{A}_G$ be the PAC with membership query learning algorithm for $\mathcal{C}_G$.

4 Simulate $\mathcal{A}_S$ (with parameters $\epsilon/2$ and $\delta/2$) as follows:

5 If $\mathcal{A}_S$ requests an example, then draw a random labeled example $(x, f(x))$ from D.

6 If $\mathcal{A}_S$ performs a membership query **member**(x) then perform a membership query on x to obtain $f(x)$.

7 If $f(x) = 1$ then give $(x, 1)$ to $\mathcal{A}_S$,

8 Else give $(x, 0)$ to $\mathcal{A}_S$.

9 Let h_S be the hypothesis output by $\mathcal{A}_S$.

10 Simulate $\mathcal{A}_G$ (with parameters $\epsilon/2$ and $\delta/2$) as follows:

11 If $\mathcal{A}_G$ requests an example, then draw a random labeled example $(x, f(x))$ from D.

12 If $\mathcal{A}_G$ performs a membership query **member**(x) then perform a membership query on x to obtain $f(x)$.

13 If $f(x) = 1$ or $f(x) =$ "?" then give $(x, 1)$ to $\mathcal{A}_G$,

14 Else give $(x, 0)$ to $\mathcal{A}_G$.

15 Let h_G be the hypothesis output by $\mathcal{A}_G$.

16 Return the hypothesis $\text{Agree}_{\{h_S, h_G\}}$.

Figure 5.1: A method for learning the agreement of nested concepts.

negative counterexample to h_S, otherwise it must be a positive counterexample for h_G. Positive and negative examples returned from an equivalence query are handled similarly.

Another modification of the above algorithm produces an on-line learning form of Theorem 58. More specifically, the modified algorithm receives an arbitrary sequence of examples x, and is asked to predict $f_?(x)$. The algorithm may ask membership queries (about examples $y \neq x$). Then it predicts the value of $f_?(x)$. The modified algorithm spends only polynomial time to output each next prediction, and the number of times it will predict incorrectly, regardless of the sequence of trials, is bounded by a polynomial in n, $|f_s|$ and $|f_g|$.

5.4.2 A General Technique for Learning Agreements

We now give a general technique that allows Theorem 58 to be applied to arbitrary blurry concepts. We show how an arbitrary agreement of concepts from a class $\mathcal{C}$ (and hence, an arbitrary blurry concept $f_?$ from $\mathcal{C}_?$), can be represented, without significant increase in size, as the agreement of two nested concepts, one of which is an intersection of concepts from $\mathcal{C}$, and the other is a union of concepts from $\mathcal{C}$.[3] Thus when unions and intersections of concepts from $\mathcal{C}$ are learnable, then the blurry class $\mathcal{C}_?$ will be learnable. We then apply this technique to show that, with some restrictions, the class of monomials, monotone DNF formulas, and DNF formulas with a constant number of terms are learnable from a consistently ignorant teacher.

Definition 59 *Let F be a finite set of boolean functions. The function Union_F is a boolean function which classifies example x as* "1" *if $f(x) = 1$ for some $f \in F$ and as* "0" *otherwise. Likewise, Intersect_F classifies x as* "1" *if $f(x) = 1$ for each $f \in F$ and as* "0" *otherwise.*

To discuss algorithms for learning unions or intersections of concepts from a given class, we must provide a size measure for each concept in the class. As we defined $|\text{Agree}_F|$, we define $|\text{Union}_F|$ (respectively, $|\text{Intersect}_F|$) to be the minimum, taken over all $F' \subseteq \mathcal{C}$ for which $\text{Union}_{F'} = \text{Union}_F$ (respectively, $\text{Intersect}_{F'} = \text{Intersect}_F$), of the representation size of F'. Each concept Agree_F in the class $\mathcal{C}_?$ is equivalent to the agreement of two concepts – Union_F and Intersect_F, and the size of this alternate representation is at most twice the size of the original representation.

[3] There is an interesting relationship, discussed in Section 5.6, between this observation and Mitchell's version space algorithm [80].

Lemma 60 *If F is a finite subset of concept class $\mathcal{C}$, then $Agree_F = Agree_{\{Intersect_F, Union_F\}}$, and the size of $Agree_{\{Intersect_F, Union_F\}}$ is at most twice the size of $Agree_F$.*

Proof: The assertion about size holds by definition, as for both functions the determining value is the size of the set F, that appears once for one function, and twice for the other.

The functions are equivalent: note that for any example x, $\text{Agree}_{\{\text{Intersect}_F, \text{Union}_F\}}(x) = 1$ if and only if $\text{Intersect}_F(x) = 1$ since $\text{Intersect}_F(x) \leq \text{Union}_F(x)$. It therefore follows that $\text{Agree}_{\{\text{Intersect}_F, \text{Union}_F\}}(x) = 1$ if and only if for all $f \in F$, $f(x) = 1$. But, by definition, this is exactly when $\text{Agree}_F(x) = 1$. An analogous argument shows that the two functions are identical when x is a negative example. Finally, since they agree as to which examples are classified as positive and negative, they must also agree as to which examples are to be classified "?". □

We now use this characterization to obtain an algorithm for learning from a consistently ignorant teacher when finite sets of unions and intersections from the given class are known to be learnable. To aid the exposition, we introduce $\mathcal{C}_\cap$ to denote the set $\{\text{Intersect}_F : F \text{ a finite subset of } \mathcal{C}\}$, and $\mathcal{C}_\cup$ to denote $\{\text{Union}_F : F \text{ a finite subset of } \mathcal{C}\}$.

Theorem 61 *Let $\mathcal{C}$ be a concept class for which $\mathcal{C}_\cap$ and $\mathcal{C}_\cup$ are PAC with membership query (respectively PAC) learnable. Then $\mathcal{C}_?$ is PAC with membership query (respectively PAC) learnable.*

Proof: For any collection F of concepts, $\text{Intersect}_F \subseteq \text{Union}_F$, and thus by Theorem 58, if $\mathcal{C}_\cap$ and $\mathcal{C}_\cup$ are PAC with membership query learnable then so is $\{\text{Agree}_{\{\text{Intersect}_F, \text{Union}_F\}} : F \subseteq \mathcal{C}\}$. Combined with Lemma 60 we get the desired result. As in Theorem 58 this result also applies when no membership queries are provided to the learner. □

The following corollary shows that the agreement of monomials with nonempty intersection is learnable.

Corollary 62 *Let $\mathcal{C}$ be the class of monomials, and let $\mathcal{C}_?^+ = \{c : c \in \mathcal{C}_?, \exists x\ c(x) = 1\}$. Then $\mathcal{C}_?^+$ is PAC with membership query learnable.*

Proof: The class $\mathcal{C}_\cap$ is learnable since $\mathcal{C}$ is closed under intersection and known to be learnable [98]. If $F \subseteq \mathcal{C}$ is a subset for which there is some example x such that $\text{Intersect}_F(x) = 1$,

then x satisfies every monomial in F and so it cannot be the case that some variable appears both negated and unnegated in F. Thus Union_F is a unate DNF formula, that is PAC with membership query learnable [9]. It is also easily verified that for any finite $F \subseteq \mathcal{C}$, the size of the representation of Intersect_F as a monomial and the size of the representation of Union_F as a unate DNF formula are each $O(\sum_{f \in F} |f|)$. Thus by Theorem 61, $\mathcal{C}_?^+$ is learnable. □

Note that in the above corollary, had we not thrown out those blurry concepts of $\mathcal{C}_?$ for which there were no positive examples, our proof would fail because it may not be the case that Union_F is unate. Thus, we would face the task of learning the class of arbitrary DNF formulas. Also note that the corollary applies to the dual of monomials (i.e. 1-DNF) when we discard blurry concepts that have no negative examples.

We next consider agreements, unions, and intersections, of at most a constant number k of concepts from a class $\mathcal{C}$. Let $\mathcal{C}_?(k) = \{\text{Agree}_F : F \subseteq \mathcal{C}, |F| \leq k\}$, $\mathcal{C}_\cap(k) = \{\text{Intersect}_F : F \subseteq \mathcal{C}, |F| \leq k\}$, and $\mathcal{C}_\cup(k) = \{\text{Union}_F : F \subseteq \mathcal{C}, |F| \leq k\}$. Applying the known learning results for monotone DNF [98] and DNF formulas with a constant number of terms [1, 18, 21] and the known learning results for decision trees [29] and DFAs [3], we obtain the following corollary. (The corollary follows because the intersection and union of a constant number of concepts from each of the preceding classes can be represented by a single concept in the corresponding class that is at most polynomially larger.)

Corollary 63 *Let $\mathcal{C}$ be the class of monotone DNF (ℓ-term DNF, decision trees, DFAs) formulas. Then for each constant k, $\mathcal{C}_?(k)$ is PAC with membership query learnable. The dual results for monotone CNF formulas and ℓ-clause CNF formulas also hold. In the case of decision trees, the hypothesis space is conjunctions of unate DNF.*

5.4.3 Learning Unions of Boxes in Euclidean Space

We now turn our attention to geometric concepts. In this section we give an algorithm to learn the agreement of a set of s axis-parallel boxes in d-dimensional Euclidean space (E^d) when the set of boxes have a samplable intersection. Throughout the remainder of this section, unless otherwise specified, by a box we mean an axis-parallel box in E^d. It is easy to show that this class is a generalization of unate disjunctive normal form formulas, and a specialization of the class of unions of boxes in E^d.

Previous algorithms that PAC learn an s-fold union of boxes in E^d include: Long and Warmuth [75] that runs in time polynomial in d only and Blumer et al. [23] that runs in time polynomial in s only. There has also been work on learning unions of s boxes in the discretized space $\{1,\ldots,n\}^d$. Most of this work has focused on the special case in which $d = 2$. Chen and and Maass [32] gave an algorithm to learn the union of two axis-parallel rectangles in the discretized space $\{1,\ldots,n\} \times \{1,\ldots,m\}$ in time polynomial in $\log n$ and $\log m$, where one rectangle has a corner in the top left corner and the other has a corner in the bottom right corner. While learning the *union* of these two rectangles within these time bounds was difficult, learning the *agreement* of the rectangles is quite simple since the learner needs only learn the intersection of the two rectangles which is easily achieved.

Chen [30] gave an algorithm that uses $O(\log^2 n)$ equivalence queries to learn the union of two rectangles in the discretized plane (i.e. $\{1,\ldots,n\}^2$). Also, Chen and Homer [31] have given an algorithm to learn the union of s rectangles in the discretized plane using $O(s^3 \log n)$ membership and equivalence queries and $O(s^5 \log n)$ time. More recently, Goldberg, Goldman and Mathias [49] have given an algorithm to learn the union of s discretized boxes in $\{1,\ldots,n\}^d$ that makes at most $sd+1$ equivalence queries and uses $O((8s)^d + sd \lg n)$ time and membership queries. Note that their algorithm, like the PAC algorithm of Long and Warmuth [75], only runs in polynomial time for d constant.

The algorithm we present here is a PAC with membership query learning algorithm for the *agreement* of s boxes in E^d that runs in time polynomial in $1/\epsilon$, $1/\delta$, s, and 9^d. Thus, the algorithm runs in polynomial time without demanding that one of s and d be constant (e.g. d can be $O(\log s)$).A key algorithm we use in learning the agreement of boxes is a PAC with membership query learning algorithm for the union of a set of boxes that all lie in the same quadrant of E^d, and for which the intersection region contains the origin. We call each box in such a set an *origin-incident* box because each such box touches the coordinate axes in such a way that the box contains the origin as a corner. Our algorithm to learn the union of s origin-incident boxes runs in time polynomial in both d and s.

To aid in learning the agreement of boxes, we also use the known algorithm for computing the intersection of boxes [23]. Namely, we first learn an approximation for the intersection region by applying the standard algorithm with all "?" examples treated as negative. Since the boxes have a non-empty intersection, we can subdivide E^d into at most 3^d sub-regions based

on this common intersection. Each sub-region can be translated and relabeled so that we can apply our algorithm for learning the union of origin-incident boxes. In the worst case, some piece of each of the s boxes will lie in each of the 3^d regions of the sub-divided problem forcing us to learn $O(s3^d)$ boxes.

It is important to note that in obtaining our algorithm to learn the agreement of boxes we take advantage of our ability to efficiently compute the intersection region and then use this information to aid in more efficiently learning the union of the boxes. It is uncommon for *both* intersections and unions of concepts to be learnable, and thus, the possibility that information from one of these could be used to learn the other is of particular interest.

5.4.3.1 Learning the Union of Origin-incident Boxes

We present an algorithm to learn the union of s origin-incident boxes in E^d where all of the boxes are in the same quadrant (for simplicity we only present the algorithm for the positive quadrant). We refer to the class of origin-incident boxes in the positive quadrant as BPQ. We define the *upper* corner of a box $b \in$ BPQ to be the corner of the box diametrically opposed to the origin. Since any box in BPQ is uniquely identified by its upper corner, we denote an origin-incident box by box(p) where p is its upper corner. Finally, we define *maxCorner* to be a function that takes a set of points in the positive quadrant of E^d and returns the upper corner of the smallest box in BPQ that contains every point in the set.

In the PAC model, an important contribution in characterizing what concept classes are learnable was made by Blumer et al. [23]. A finite set $S \subseteq \mathcal{X}$ is shattered by $\mathcal{C}$ if for each subset $S' \subseteq S$, there is a concept $f \in \mathcal{C}$ that contains all of S' and none of $S - S'$. The *Vapnik-Chervonenkis dimension* of $\mathcal{C}$, denoted $\text{VCD}(\mathcal{C})$, is defined to be the largest d for which some set of d points is shattered by $\mathcal{C}$. Building on the work of Vapnik and Chervonenkis [101], Blumer et al. proved that any PAC learning algorithm must draw at least $\Omega(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(\mathcal{C})}{\epsilon})$ examples. Furthermore, they proved that the general technique of finding a hypothesis consistent with a set of $\Theta(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCDC}}{\epsilon}\ln\frac{1}{\epsilon})$ examples, when feasible, always results in a (possibly super-polynomial time) PAC learning algorithm[4]. We use this result to show that $\text{BPQ}_\cup(s)$ is PAC with membership query learnable.

[4]There are some technical restrictions on the concept classes for which this result applies.

```
LearnBPQ(S)
    /* S is a labeled sample. */
    /* This algorithm will, with probability at least 1 − δ, output a hypothesis with error
    at most ε given that |S| ≥ max{(4/ε) log(2/δ), (16ds log 3s/ε) log(13/ε)}. */
1   h := ∅ /* The set of boxes in the hypothesis; represented as upper corners */
2   P := {x : x ∈ S, x is a positive example}
3   while there exists an example x ∈ P
4       P := P − {x}
5       for each y ∈ P if member(maxCorner{x, y}) = "yes" then
6           x := maxCorner{x, y}
7           P := P − {y}
8       add box(x) to h
9   return h /* That is, output the union of boxes in h*/
```

Figure 5.2: Algorithm to learn a union of origin-incident boxes.

Theorem 64 *Let $BPQ_\cup(s)$ be the union of at most s origin incident boxes. The class $BPQ_\cup(s)$ is PAC with membership query learnable with time and sample complexity polynomial in s, d, $1/\epsilon$, $1/\delta$.*

Proof: To prove the theorem, we show that

1. Algorithm LearnBOQ (Figure 5.2), takes as input a sample S, runs in time polynomial in S, and outputs a union of at most s origin-incident boxes (that is, an element of $\mathrm{BPQ}_\cup(s)$) that is consistent with the sample.

2. The VC-dimension of $\mathrm{BPQ}_\cup(s)$ grows polynomially with s and d (in particular, it is at most $2ds\log 3s$).

It then follows from Theorem 2.1 of Blumer et al. [23] that if LearnBOQ is given a sample of cardinality at least $m = \max\left\{\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{16ds\log(3s)}{\epsilon}\log\frac{13}{\epsilon}\right\}$, then with probability at least $1-\delta$, it will output a hypothesis h with error at most ϵ.

To see that (2) is true, we that the VC-dimension of BPQ is at most d. A point can be uniquely excluded from a set of points by a box in BPQ box only if at least one of its components is the largest among all points in the set for that component; since there are only d

components, the VC-dimension can be at most d.[5] Then by Lemma 3.2.3 of Blumer et al. [23], the VC-dimension of $\mathrm{BPQ}_{\cup}(s)$ is at most $2ds\log(3s)$. To complete the proof, it remains to be shown that (1) holds. We first show that LearnBOQ produces a hypothesis that is consistent with the sample S. The hypothesis produced is consistent with the positive examples of S since the algorithm does not terminate until all positive examples of S have been removed from P and no point is removed unless the box about to be placed in h contains it. Furthermore, if $\mathrm{box}(x)$ was placed in h, then x was a positive example (either it was in P or verified to be positive with a membership query). Since x is a positive example, $\mathrm{box}(x)$ is contained within some box of the target. Thus no negative points (even those not in S) can be contained in any of the boxes placed in h.

We now prove that the hypothesis h output by LearnBOQ contains at most s boxes. Suppose h contained more than s boxes. Since each box of h is contained within a box of the target, it follows that there must be at least two boxes (say b_i and b_j) in h that are contained within the same box (say b_k^*) of the target. Assume, without loss of generality, that b_i was placed in h first. Let p_i be the point from P selected in step 3 during the iteration of the while loop in which b_i was added to h. Thus p_i must be contained within b_i. Likewise, let p_j be the point from P selected during the iteration of the while loop in which b_j was added to h. (So p_j is in b_j.) Since $p_j \in P$ after b_i was placed in h, a membership query must have been performed on $\mathrm{maxCorner}\{p_i', p_j\}$, where $\mathrm{box}(p_i')$ contains p_i. Furthermore, since p_j was not removed during the construction of b_i, it follows that $\mathrm{maxCorner}\{p_i', p_j\}$ is a negative example. Since p_i is contained within $\mathrm{box}(p_i')$ it must be that $\mathrm{maxCorner}\{p_i, p_j\}$ is also a negative example. Recall that the box b_k^* of the target contains b_i and b_j and thus b_k^* contains p_i and p_j. However, this contradicts the fact that $\mathrm{maxCorner}\{p_i, p_j\}$ is a negative example. Thus h contains at most s boxes.

Finally, note that LearnBOQ runs in polynomial time, since there are at most s iterations of the while loop, each of which takes at most $O(m)$ time, where m is the cardinality of S. This completes the proof of (1) above, and hence of the theorem.

□

[5] In fact, the VC-dimension is exactly d because any subset S of the set of points $\{p_i : p_i = \langle x_1, \ldots, x_d\rangle, x_i = 3, x_{j(\neq i)} = 1\}$ can be selectively excluded from the set by the box $b \in \mathrm{BPQ}$ having upper corner $\langle x_1, \ldots, x_d\rangle$ where $x_{i \in S} = 2$ and $x_{i \notin S} = 4$.

We note that LearnBOQ is easily modified to obtain an algorithm to learn the union of s origin-incident boxes in E^d when all of the boxes are in any single quadrant.

5.4.3.2 Learning the Agreement of Boxes with Samplable Intersection

In this section we give an algorithm to learn the agreement of s boxes in E^d (hence, an algorithm to learn boxes from a consistently ignorant teacher) when the intersection region is samplable. Our algorithm has polynomial time and sample complexity in both d and s when $d = O(\log s)$. The intuition behind our algorithm lies in the way in which the non-empty intersection of a set of boxes can be used to partition E^d into 3^d sub-regions. Let B be the set of boxes for which we are computing the agreement. Figure 5.3 illustrates the effect of this partitioning on a typical box $b \in B$. The large, transparent box is b, and the solidly shaded box b_I in the center is the intersection of all boxes in B (and thus must be contained in b). By infinitely extending the faces of b_I we decompose b into a set of sub-boxes that are also axis-parallel. A few of the sub-boxes generated by this extension have been cross-hatched. Notice that some of the sub-boxes generated in this manner share a face with b_I, others share only an edge, and still others share only a corner. Including b_I itself, the decomposition of E^3 results in $3^3 = 27$ sub-regions. In general, there are 3^d sub-regions in E^d as seen informally by first observing that the bounds of the intersection region are d pairs of parallel hyperplanes, one pair of parallel hyperplanes for each dimension. Thus, in each dimension the sub-region lies either above both of the hyperplanes, lies between the pair, or lies beneath both of the hyperplanes. Therefore, we have 3 choices for each set of bounding planes giving a total of 3^d sub-regions.

Sub-regions and Sub-boxes. A useful way to categorize these 3^d sub-regions is by the dimension of the boundaries they share with the intersection region. For example, in Figure 5.3 a sub-region that shares a face with the intersection region shares a 2-dimensional boundary. A sub-region that shares an edge shares a 1-dimensional boundary, and a sub-region that shares only a corner shares a 0-dimensional boundary.

Since, by definition, the intersection region is contained in every box in B, the manner in which these boxes can overlap in a sub-region is restricted based on the dimension of the boundary that the sub-region shares with the intersection region. Figure 5.4 illustrates the constraints imposed by the dimension of the shared boundary—the higher the dimension of the

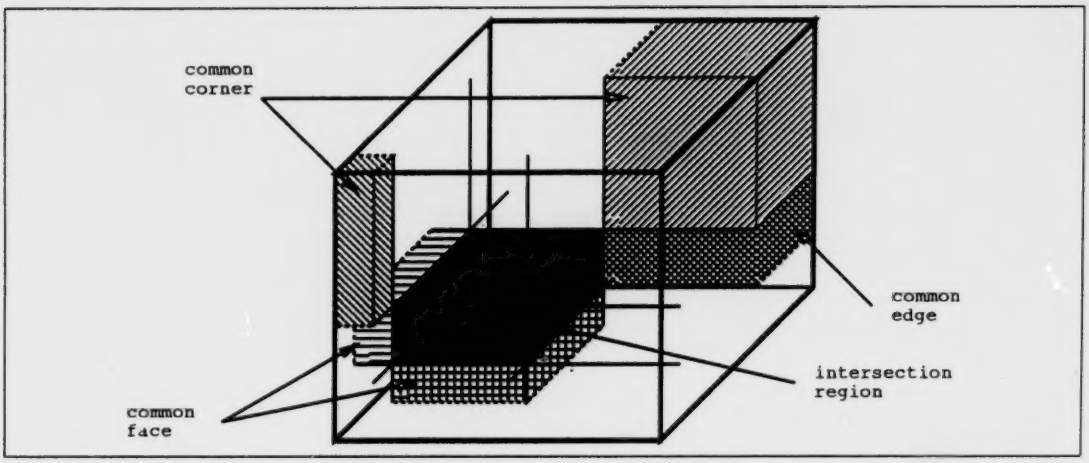

Figure 5.3: Decomposition of an axis-parallel box with respect to the intersection region.

shared boundary the greater the number of dimensions that are constrained by the intersection. From the figure we see that in sub-regions that share only a corner point with the intersection region, we have almost no information about the way the sub-boxes in that sub-region might be arranged. At the other extreme, in the sub-regions that share a face with the intersection region, we know almost precisely how the sub-boxes are arranged—the only unknown is how far beyond the face the sub-boxes extend.

To eliminate the dimensions of a sub-region that are already constrained by the intersection region we introduce the following notation. Let $p = \langle x_1, x_2, \ldots, x_d \rangle$ be a point in E^d, and let I be a set of indices $\{i_1, i_2, \ldots, i_k\}$ such that $1 \leq i_1 < i_2 < \ldots < i_k \leq d$. Then the point $\pi_I(p) = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \rangle$ in E^k is the *projection* of p with respect to I. In general, if a sub-region shares a k-dimensional boundary with a d-dimensional intersection region, then for any sub-box in that sub-region we need only determine the sub-box's extent in the remaining $d-k$ dimensions. Equivalently, boxes in the same sub-region can be translated to all be origin-incident boxes in a $d-k$ dimensional space for which we can apply LearnBOQ. Furthermore these $d-k$ dimensions are exactly those which are not shared with the intersection region. Let the intersection region of the boxes in B be $\{\langle x_1, x_2, \ldots, x_d \rangle : w_i \leq x_i \leq z_i\}$ for constants w_i and z_i $(1 \leq i \leq d)$. Now for any point $p = \langle a_1, a_2, \ldots, a_d \rangle \in E^d$, if $w_i \leq \pi_i(p) \leq z_i$ and p is contained in some box $b \in B$ then for any point y such that $w_i \leq y \leq z_i$, the point $\langle a_1, a_2, \ldots, a_{t-1}, y, a_{t+1}, \ldots, a_d \rangle$ is also contained in b. Thus for any point p we can ignore any dimension i of p if p lies in a sub-region that is bounded between the pair of parallel hyperplanes

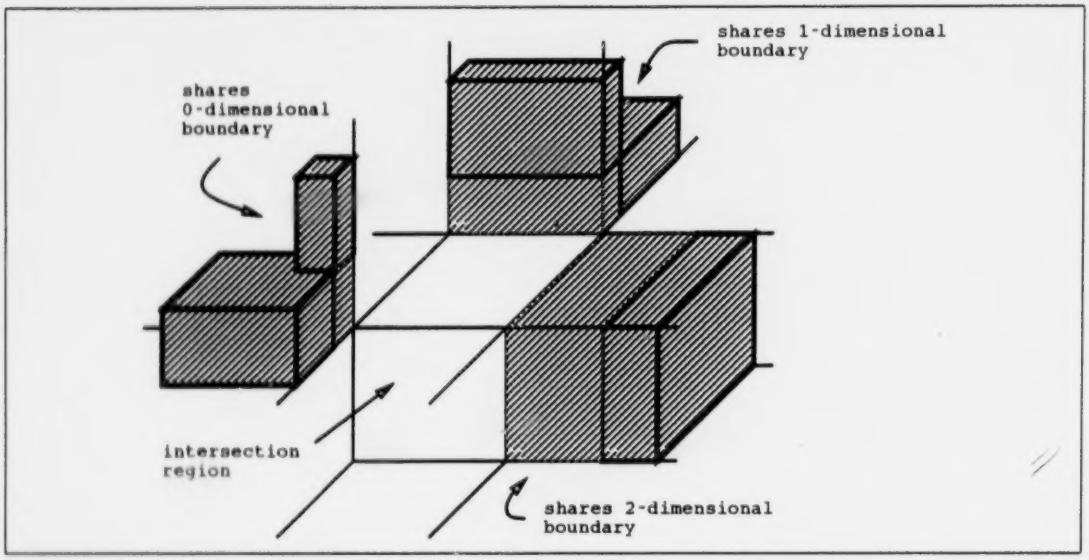

Figure 5.4: Sub-region constraints imposed by the dimension of the boundary shared with the intersection region.

that bound dimension i of the intersection region. This observation is used by our algorithm to take a collection of points for a specific sub-region, project out any dimension for which the sub-region is bounded, and then learn all the sub-boxes in that sub-region using a version of LearnBOQ of suitable dimension for the resulting projected and translated points.

Removing Intersection Box Estimation Error. There remains a subtle point that we must address. So far we have assumed that we know the intersection region *exactly*. However, in reality, we apply a known PAC algorithm [23] to obtain a good approximation of the intersection region; the approximation box is contained in the intersection region. To obtain an approximation with error at most ϵ with probability at least $1 - \delta$, this algorithm draws a sample of size $\max\left(\frac{4}{\epsilon}\log\frac{2}{\delta}, \frac{16d}{\epsilon}\log\frac{13}{\epsilon}\right)$ and returns the smallest box that is consistent with the sample. Let IBox(S) be a procedure that takes a sample S and returns the smallest box consistent with S. To apply this algorithm to learn the intersection region of the boxes in our model, we simply modify the sample by changing all "?" examples to negative examples.

The difficulty presented here is that the sub-region in which a point p lies may differ when subdividing based on the true intersection region versus sub-dividing based on the underestimate for the intersection region. Figure 5.5 illustrates how this may happen where A^* is the true

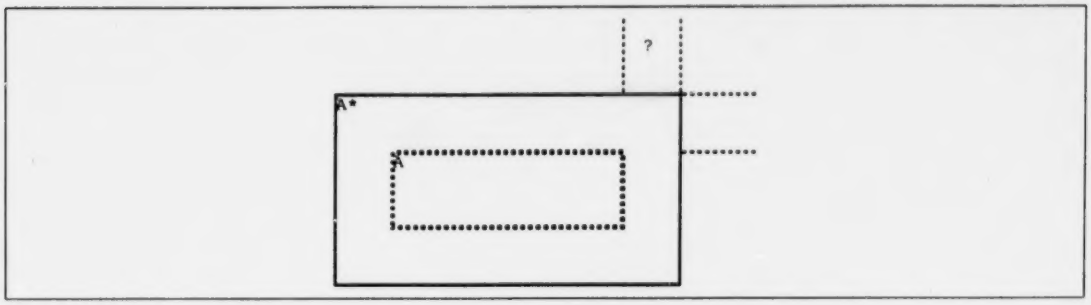

Figure 5.5: An example assigned to the wrong sub-region.

intersection region and A is our (underestimate) of A^*; the point marked "?" lies between the vertical boundaries for A^*, but lies to the right of the vertical boundaries for A. We handle this difficulty by discretizing E^d with an irregular Cartesian grid.

Suppose we have a collection S of points from E^d. For each dimension i consider the set $S_i = \{\pi_{\{i\}}(p) : p \in S\}$. Notice that S_i is a collection of points from E^1 and that if we consider labeling the coordinate axis for dimension i of E^d using only values found in S_i, then we will have effectively discretized E^d in such a way that *every* point of the sample S lies at some intersection point of the resulting irregular Cartesian grid. We then expand our estimate A of the true intersection region A^* in such a way that for every point p in S, the sub-region generated by A in which p lies and the sub-region generated by A^* in which p lies are the same. An algorithm to achieve this goal is given in Figure 5.6. We have the following lemma.

Lemma 65 *Let A be a non-empty underestimate of the true intersection region A^*. The algorithm* **Expand**(A, S) *outputs a box A' so that for all $p \in S$, the sub-region generated by A' in which p lies and the sub-region generated by A^* in which p lies are the same. Furthermore,* Expand *runs in time polynomial in the size of S.*

Proof Sketch: Consider the infinitely tall, finitely wide strip in Figure 5.5 bounded between the right edges of A and A^* in which the "?" point lies. If this strip were samplable, **Expand** would have been likely to witness some point in the strip. Any point in the strip, when projected between the top and bottom edges of A would have been a positive example. Thus the right edge of A would have been closer to the right edge of A^*. □

Expand(A,S)

/* A is a non-empty underestimate of the true intersection region. */

/* S is a set of labeled example points. */

1 Let $\langle a_1, a_2, \ldots, a_d \rangle$ be any point contained in A.

2 For $i := 1$ to d

3 /* Use membership queries to obtain a lower bound for dimension i. */
Set ℓ_i to be the smallest v such that $v = \pi_{\{i\}}(p)$ for some $p \in S$ and $\langle a_1, a_2, \ldots, a_{i-1}, v, a_{i+1}, \ldots, a_d \rangle$ is a positive example.

4 /* Use membership queries to obtain an upper bound for dimension i. */
Set u_i to be the largest v such that $v = \pi_{\{i\}}(p)$ for some $p \in S$ and $\langle a_1, a_2, \ldots, a_{i-1}, v, a_{i+1}, \ldots, a_d \rangle$ is a positive example.

5 Return the box A' having opposing corners $\langle \ell_1, \ell_2, \ldots, \ell_d \rangle$ and $\langle u_1, u_2, \ldots, u_d \rangle$.

Figure 5.6: Algorithm to expand an underestimate A of the the true intersection region A^* to an estimate A' such that for any point p in S, the sub-region generated by A' in which p lies is the same as the sub-region generated by A^* in which p lies.

The Full Algorithm. Putting all the pieces together we obtain our algorithm. We first approximate the intersection region, and then we refine this estimate using Expand. Next we apply a version of LearnBOQ of suitable dimension to the points in the various sub-regions generated by the intersection region. Finally, we combine the hypotheses obtained from the calls to LearnBOQ along with our estimate of the intersection region to obtain our final hypothesis. The complete algorithm is shown in Figure 5.7.

Theorem 66 *LearnBoxesAgreement is a PAC with membership query algorithm for learning the agreement of s axis-parallel boxes in E^d. Let p^+ be the probability of receiving a positive example from D. The sample complexity is $m = O\left(\frac{9^d}{\epsilon^2}\log\frac{3^d}{\delta} + \frac{9^d}{\epsilon^2}ds\log s\log\frac{3^d}{\epsilon} + \frac{1}{p^+}\log\frac{1}{\delta}\right)$, and the time complexity is $O(sm)$.*

Proof: Note that by drawing a sample of size $1/p^+ \ln 2/\delta$ with probability at least $1 - \delta/2$ we will obtain a positive example. It follows directly from Blumer et al. [23] that our sample suffices to ensure that the hypothesis output by IBox(T) has error at most $\epsilon/3^d$ with probability at least $1 - \frac{\delta}{2\cdot 3^d}$. We now show that for each of the $3^d - 1$ remaining sub-regions, the sample is sufficiently large so that with probability at least $1 - \frac{\delta}{2\cdot 3^d}$ there are enough points so that the hypothesis output by LearnBOQ for that region has error at most $\epsilon/3^d$. It immediately follows from Theorem 64 that it is sufficient to provide LearnBOQ with a

LearnBoxesAgreement()

1 Draw a sample S of size $m := \max\left\{\frac{8 \cdot 9^d}{\epsilon^2} \lg \frac{4 \cdot 3^d}{\delta}, \frac{32 \cdot 9^d \cdot d s \lg(3s)}{\epsilon^2} \lg \frac{13 \cdot 3^d}{\epsilon}, \frac{1}{p^+} \lg \frac{1}{\delta}\right\}$.

2 If there are no positive examples halt and report failure.

3 Let T be set of examples obtained by relabeling all "?" examples of S as negative.

4 Set $A := \text{Expand}(\text{IBox}(T, S))$.

5 Let R be the set of sub-regions generated by A (excluding A itself).

6 For each sub-region $r \in R$

7 Choose any point p_r in the boundary shared by A and the sub-region r such that p_r is an extreme point in every dimension of the boundary. Let f_r be the coordinate transformation that translates f_r to the origin of E^d.

8 /* Identify dimensions for which we already know the extent of any sub-box lying in r */
Let I_r be the dimensions for which r is not bounded between a pair of parallel hyperplane bounds for A.

9 /* Project out those dimensions for any point of S that lies in r, relabel "?" examples */
Let $S_r := \{p' : p \in r \text{ and } p' = \pi_{I_r}(f_r(p))\}$.
If $p' \in S_r$ is labeled with "?" then relabel it as positive.

10 Set B_r to be the set of boxes returned by LearnBOQ(S_r).

11 Given any unlabeled example x, predict

$$\begin{cases} 1 & \text{if } x \text{ lies in } A \\ ? & \text{if } \exists r \in R, b \in B_r \text{ such that } x \text{ lies in sub-region } r \text{ and } \pi_{I_r}(f_r(x)) \text{ lies in } b \\ 0 & \text{otherwise} \end{cases}$$

Figure 5.7: The algorithm LearnBoxesAgreement for learning the agreement of a set of axis-parallel boxes with samplable intersection region.

sample of $m' = \max\left\{\frac{4\cdot 3^d}{\epsilon}\log\frac{4\cdot 3^d}{\delta}, \frac{16\cdot 3^d ds\log(3s)}{\epsilon}\log\frac{13\cdot 3^d}{\epsilon}\right\}$ points from the given region. By applying Chernoff bounds from Fact 6 it is straightforward to show that a sample of size $m = \max\left\{\frac{2}{\epsilon}m'3^d, \frac{8\cdot 3^d}{\epsilon}\ln\frac{2\cdot 3^d}{\delta}\right\}$ is sufficiently large so that there for each sub-region of weight at least $\frac{\epsilon}{3^d}$ there are at least m' points from the sample with probability at least $1-\frac{\delta}{2\cdot 3^d}$. (Note that sub-regions with weight less than $\epsilon/3^d$ can contribute at most $\epsilon/3^d$ to the total error.) It follows from Lemma 65 that the total error of our final hypothesis is the sum of the errors of the hypotheses we generate for each of the 3^d regions. Thus the probability that the error of the final hypothesis is more than $3^d\cdot\epsilon/3^d = \epsilon$ is at most $3^d\frac{\delta}{2\cdot 3^d} = \delta/2$.

We now compute the time complexity. It is easily argued that the first five steps take $O(m)$ time. Observe that the loop in step 6 goes over $3^d - 1$ regions. While each of the s boxes can be sub-divided to have a piece in each sub-region, each point in the sample falls into at most one of the regions. Thus the total time for step 6 is $\sum_{r\in R} O(sm_r)$ where m_r is the number of sample points that are in region r. Finally since $\sum_{r\in R} m_r \leq m$, we get that the total time for step 6 is at most $O(sm)$. □

5.5 A Negative Result

By the results of chapter 2, the class of propositional Horn sentences is known to be PAC with membership query learnable. We provide evidence that this result cannot be strengthened to allow learning Horn sentences from a consistently ignorant teacher, by showing that such an algorithm could be used to learn the class of DNF formulas.

Unlike in the previous section where we used the knowledge of the intersection region of a set of s boxes to aid in an algorithm to learn the agreement of any set of s boxes having a non-empty intersection, here we show that the knowledge of the intersection region of a set of Horn sentences appears to be insufficient in providing the leverage needed to learn the disjunction (and thus agreement) of the Horn sentences.

Let DHF represent the class of disjunctions of Horn Sentences. We begin with the observation that the class of DNF formulas is a subset of the class of DHF formulas.

Claim 67 *For any DNF formula there exists a logically equivalent DHF formula.*

Proof: Observe that every unnegated literal v is equivalent to the Horn clause $(T \rightarrow v)$ and every negated literal $\bar{v}$ is equivalent to the Horn clause $(v \rightarrow F)$. For example, $a\bar{b}c \equiv (T \rightarrow a)(b \rightarrow F)(T \rightarrow c)$. Thus we can represent each term by a Horn sentence and take the disjunction of these Horn sentences to build a DHF formula that is logically equivalent to the given DNF formula. Finally, observe that the size of the DHF formula created by this transformation has size polynomial in the DNF formula from which it was created. □

Using the above observation, it is easily shown that the problem of learning an agreement of Horn sentences (without any restrictions) is as hard as learning DNF. However, as demonstrated by our algorithm to learn the agreement of boxes, if the intersection of the Horn sentences in the agreement were non-empty then it may be possible to use the intersection information to successfully learn the disjunction. We now prove the stronger negative result that learning the agreement of Horn sentences even when the intersection region is samplable is as hard as learning the class of DNF formulas.

Theorem 68 *PAC with membership query learning the agreement of Horn sentences for which the intersection region is samplable is as hard as PAC with membership query learning the class of DNF formulas.*

Proof: We prove this through a sequence of prediction preserving reductions [86]. Let DHF-1pos be the class of DHF formulas with exactly one positive example p that satisfies every disjunct. Let agree-Horn-1pos be the agreement of Horn sentences that have exactly one example in their intersection. Finally, let agree-Horn be the agreement of Horn sentences with a samplable intersection region.

Applying Claim 67 it immediately follows that the learnability of DHF implies the learnability of DNF. We now give a reduction showing that the learnability of DHF-1pos (even when the learner has a priori knowledge of the single positive example) implies the learnability of DHF . Let f be the target of the algorithm for DHF and f_p be the target of the algorithm to be constructed for DHF-1pos . Choose as the zero vector as the positive example p known to the learner, and let $\{v_1, ..., v_n\}$ be the set of variables over which f is defined. We construct f_p from f by adding an extra literal v to the antecedent of every Horn clause in f as well as adding the Horn sentence $(v \rightarrow F)(v_1 \rightarrow F) \cdots (v_n \rightarrow F)$. Note that the only example satisfying every disjunct of f_p is the zero vector p.

The DHF algorithm $\mathcal{A}$ simulates the queries for the DHF-1pos algorithm $\mathcal{A}_+$ as follows: When $\mathcal{A}_+$ requests an example, $\mathcal{A}$ obtains a random example x (that assigns values only to $v_1, \ldots, v_n$), generates the example x' that is like x with the additional variable v set to 1, and gives x' to $\mathcal{A}_+$ labeled as x was. If $\mathcal{A}_+$ makes a membership query on some vector x', if $v = 0$ then $\mathcal{A}$ returns "1" and if $v = 1$ then $\mathcal{A}$ responds with the result of a membership query on the example x that is just x' with the setting for the variable v eliminated. Once $\mathcal{A}_+$ terminates with hypothesis h_+, $\mathcal{A}$ is able to predict the label of any example, x, by setting v to 1 and evaluating f_+ on that example. Note that setting v to 1 causes the added Horn sentence in f_p to evaluate to 0 and the antecedents of all the remaining Horn clauses to not be affected by x.

We now show that the learnability of agree-Horn-1pos implies that DHF-1pos is learnable. It is at this point that we switch from learning a standard boolean concept to learning an agreement. Note that the learning problem for the class DHF-1pos assumes that the learner knows the single positive example that satisfies every disjunct of the the target. Any algorithm for agree-Horn-1pos can be used to learn DHF-1pos by simply providing the sole positive example of agree-Horn-1pos to DHF-1pos as p and changing all "?" examples to positive examples.

Finally, we show that the learnability of agree-Horn implies that agree-Horn-1pos is learnable. Recall that a PAC with membership query learning algorithm must learn under any distribution D. When the agree-Horn algorithm requests a random example, the simulation algorithm flips a fair coin. With probability 1/2, the simulation provides the agree-Horn algorithm with the single positive point in agree-Horn-1pos (and thus the positive region is samplable). Otherwise, a random example drawn from the oracle is given to the agree-Horn algorithm. Clearly agree-Horn is a generalization of agree-Horn-1pos and thus at least as hard.

Thus it follows from this sequence of reductions that PAC with membership query learning the agreement of Horn sentences with a samplable positive region is as hard as PAC with membership query learning the class of DNF formulas. □

Finally, we further strengthen this result by using the hardness result of Angluin and Kharitonov [10] showing that, under the assumption that one-way functions exist, membership queries do not help in learning DNF formulas (with an unbounded number of terms).

Corollary 69 *PAC with membership query learning the agreement of Horn sentences for which the intersection region is samplable is as hard as PAC learning the class of DNF formulas assuming that one-way functions exist.*

5.6 Relating Agreements and Version Spaces

We take a brief diversion and compare Mitchell's definition of version spaces [80] to agreements [78].

Given a concept class $\mathcal{C}$ of boolean functions, a finite sample E of examples, a set of positive examples $P \subseteq E$, and a set of negative[6] examples $N \subseteq E$, recall that the *version space* [80] is the set of concepts in $\mathcal{C}$ consistent with P and N. As there may be many such consistent concepts, a version space is often represented using two sets G and S defined as follows. Let

$$C_V = \{f \in \mathcal{C} : \forall n \in N \; f(n) = 0 \;\; \wedge \forall p \in P \; f(p) = 1\}.$$

Next, taking $g > f$ for functions g and f to denote that g is strictly more general than f,[7] define

$$G = \{f \in C_V : \nexists g \in C \; g > f\},$$

and define

$$S = \{f \in C_V : \nexists g \in C \; f > g\}.$$

Intuitively, G is the set of concepts in $\mathcal{C}$ consistent with a set of labeled examples such that the concepts in G are maximally general and S is the set of concepts in C_V consistent with a set of labeled examples such that the concepts in S are maximally specific.

Mitchell observes that associated with any G and S sets is a ternary function $\mathrm{VS}_{[S,G]}$ (our notation) that expresses how a version space predicts the label of an example x:

[6] It need not be the case that $P \cup N = E$.

[7] That is, for every $x \in E$, $g(x) \geq f(x)$ and there is some $x_0 \in E$ such that $g(x_0) > f(x_0)$. In other words the set of examples labeled positive by g is a proper superset of the examples labeled positive by f.

$$\mathrm{VS}_{[S,G]}(x) = \begin{cases} 1 & \text{if } s(x) = 1 \text{ for each } s \in S, \\ 0 & \text{if } g(x) = 0 \text{ for each } g \in G, \\ ? & \text{otherwise.} \end{cases}$$

We now relate agreements to $\mathrm{VS}_{[S,G]}$ and discuss why the former is more appropriate for our purposes.

Let S and G delimit a version space. To see that $\mathrm{VS}_{[S,G]}$ defines an agreement of functions, observe that for all examples x, $\mathrm{VS}_{[S,G]}(x) = \mathrm{Agree}_{S \cup G}(x)$. Specifically, if $\mathrm{VS}_{[S,G]}$ labels an example x with "?," then for some s in S and some g in G, s labels x negative, and g labels x positive. Thus, since s and g disagree on the label of x, $\mathrm{Agree}_{S \cup G}$ labels x with "?" also. If $\mathrm{VS}_{[S,G]}$ labels x positive, then, by definition, every s in S labels x positive. Since the elements of G are more general than the elements of S, every g in G labels x positive also. So $\mathrm{Agree}_{S \cup G}$ labels x positive also. (Dually when $\mathrm{VS}_{[S,G]}$ labels x negative.)

Furthermore, every agreement can be represented by a ternary function, $\mathrm{VS}_{[S,G]}$, for suitably defined S and G. Let F be a subset of $\mathcal{C}$. Recall that E is a finite subset of examples, and that Agree_F and f are functions with ranges $\{0, 1, ?\}$ and $\{0, 1\}$, respectively. Define

$$C_A = \{f \in \mathcal{C} : \forall x \in E \; f(x) = \mathrm{Agree}_F(x)\}.$$

Now define

$$G = \{f \in C_A : \nexists g \in C_A \; g > f\},$$

and

$$S = \{f \in C_A : \nexists g \in C_A \; f > g\}.$$

We show that, for all examples x, $\mathrm{Agree}_F(x) = \mathrm{VS}_{[S,G]}(x)$.

Note that all the functions in the set F are trivially contained in the set C_A. If Agree_F labels x with "?," there exists functions f_1 and f_2 in F such that f_1 labels x negative and f_2 labels x positive. Since f_1 and f_2 are themselves in F, they are also in C_A. Further, since f_1 and f_2 are in C_A, there exists s and g in S and G respectively, such that s labels x negative and g labels x positive, where s is at least as specific as f_1 and g is at least as general as f_2.

BEST COPY AVAILABLE

(So, regardless of where f_1 and f_2 "lie relative to" the elements of S and G, there is an element of S that labels x negative and an element of G that labels x positive.) Consequently, $\mathrm{VS}_{[S,G]}$ labels x with "?."

If Agree_F labels x positive then every f in C_A, labels x positive. Since S is a subset of C_A, every s in S labels x positive. Thus $\mathrm{VS}_{[S,G]}$ labels x positive also. (Dually when Agree_F labels x negative.)

We now discuss why the agreement representation is more appropriate for our purposes.

Membership Queries The concept classes we investigate (for example, the agreement of a constant number of monotone DNF, k-term DNF, or decision trees) are not known to be learnable without membership queries. It can be shown that the *candidate elimination algorithm* (CEA) (as defined in [80]) requires exponential time to learn these concept classes. However, since the CEA does not have access to membership queries, the comparison seems ill-founded. Moreover, it is not clear how to augment the CEA with membership queries. Thus, the CEA is not appropriate for the classes we consider.

Order of Labeled Examples Even if we could augment the CEA with membership queries, since the CEA is sensitive to the order labeled examples are provided, it is again not suitable for our purposes.

There has been previous evidence of the exponential growth associated with maintaining G and S sets [53, 57, 55]. Hirsh [62] has proposed avoiding this growth by constructing only the smaller of the sets G and S and maintaining the training examples as a representation of the unconstructed set. We present a situation in which the size of *both* the G and S set grow exponentially due to the order in which labeled examples are provided to the CEA. Thus, even representations which maintain only one of G or S do not inhibit the problem of exponential growth.

Let $\mathcal{C}$ be the class of monotone monomials defined over the variable set $\{x_i\}_{i=1}^{2n}$, such that $\mathcal{C}$ contains exactly the constant functions *true* and *false* together with monomials of the form $\bigwedge_{i \in S} x_i$ where S is a subset of $\{1, \ldots, 2n\}$ of cardinality exactly n. For example, if $n = 1$, $\mathcal{C} = \{false, x_1, x_2, true\}$.

Now consider how the CEA behaves for general n. Observe that initially $G = \{true\}$ and $S = \{false\}$. Suppose the following three labeled examples were supplied to the CEA:

- the positive example $\langle 1 \cdots 11 \cdots 1 \rangle$ with all variables satisfied,
- the negative example $\langle 0 \cdots 00 \cdots 0 \rangle$ with all variables falsified, and
- the negative example $\langle 1 \cdots 10 \cdots 0 \rangle$ with the first n variables satisfied and the remaining variables falsified.

After the first two examples, both G and S consist of the $\binom{2n}{n}$ monotone monomials from $\mathcal{C}$. However, altering the order of the examples so that the third example is provided first causes the G and S sets to consist of one element each. In contrast, observe that the class $\mathcal{C}$ is trivially learnable with membership queries.

Conciseness We have already demonstrated that the agreement representation is always as small as and sometimes exponentially smaller than the G and S set representation. Thus a positive result in the agreement representation is "stronger" in the sense that we are allowed possibly exponentially less time.

Why is the G and S set representation not always concise? Our purpose is only to capture a ternary function. With this in mind, one reason for this relative lack of conciseness is that the agreement representation need not use only those functions that can be partitioned into two sets, one "general" and the other "specific;" indeed the agreement representation is free to represent the ternary function as a set of mutually incomparable functions.

This relative lack of conciseness also occurs when the agreement representation does consist of functions that can be partitioned into the general and specific sets – sometimes there exist exponentially smaller subsets G' and S' of G and S respectively, that are equivalent in terms of prediction (that is, for all examples x, $\mathrm{VS}_{[G',S']}(x) = \mathrm{VS}_{[G,S]}(x)$). G and S may contain many more consistent functions than are necessary for accurate prediction [78].

5.7 Discussion

We have presented a new formal learning model for learning from a consistently ignorant teacher. Along with giving several useful characterizations of this model, we have given some general

conditions indicating when one can successfully learn under this model. In addition we present a polynomial time algorithm for learning the agreement of s boxes in E^d for $d = O(\log s)$. We have also shown that learning the agreement of Horn Sentences is as hard as learning DNF from random examples.

There are many interesting open problems raised by this work. First of all it would be interesting to explore other concept classes for which learning finite unions (or intersections) is hard to see if the agreement of concepts from the class is learnable. Also, although we have shown that the complexity of learning the agreement of an arbitrary number of Horn sentences is hard as learning the class of DNF formulas, this hardness result may not hold when the number of Horn sentences in the agreement is bounded with respect to the number of variables. For example, it is an open question as to whether or not there is an algorithm for learning the agreement of a constant number of Horn sentences.

Chapter 6

Restricted First-Order Horn Sentences

We now make a transition from propositional concepts to first-order concepts. We return to learning from entailment, and begin with a restricted class of first-order Horn sentences.

A natural framework for research in *inductive logic programming* is the investigation of the learnability/predictability of various classes of definite clause theories, particularly in the PAC [98] and exact learning models [4]. Relatively little work has been done within this framework, though interest is rising sharply [15, 34, 35, 41, 70, 81, 82]. This chapter describes new results on the learnability of several restricted classes of simple (two-clause) definite clause theories that may contain *recursive* clauses; theories with recursive clauses appear to be the most difficult to learn. The positive results are proven for learning by equivalence queries, which implies PAC learnability [4]. In obtaining the results, we introduce techniques that may be useful in studying the learnability of other classes of definite clause theories with recursion.

The results are presented with the following organization. Section 2 describes the learning model. Section 3 shows that the class $\mathcal{H}_{2,1}$, whose concepts are built from unary predicates, constants, variables, and unary functions, is learnable. Section 4 shows that the class $\mathcal{H}_{2,*}$, an extension of $\mathcal{H}_{2,1}$ that allows predicates of arbitrary arity, is not learnable under a reasonable complexity-theoretic assumption.[1] Nevertheless, Section 4 also shows that each subclass $\mathcal{H}_{2,k}$

[1] The classes $\mathcal{H}_{2,1}$ and $\mathcal{H}_{2,k}$ (discussed next) have one other restriction, that variables are *stationary*. This restriction is defined later.

of $\mathcal{H}_{2,*}$, in which predicates are restricted to arity k, is learnable *in terms of* a slightly more general class, and is therefore *PAC predictable*. The prediction algorithm is a generalization of the learning algorithm in Section 3. The results of Section 4 leave open the questions of whether (1) $\mathcal{H}_{2,*}$ is *PAC predictable* and (2) $\mathcal{H}_{2,k}$ is *PAC learnable*. Section 5 shows that the techniques used in Section 4 can also be used to prove that the class $TP^2 \cup CFTFB_{uniq}$[15], which allows higher-arity functions, cannot be learned. It conjectures that a related, and in some ways broader, class can be learned, using techniques from earlier sections. Section 6 returns to unary predicates, but allows an arbitrary number of clauses; the resulting class is called $\mathcal{H}_{*,1}$. Section 6 shows that this class is equivalent to the class of regular languages, though the question of representation size remains open. While the results of this section do not subsume any other results of the chapter, Section 6 does show that it may be possible to obtain learnability results for some classes of definite clause theories from results about formal languages and automata. Section 7 relates our results to other work on the learnability of definite clause theories in the PAC or exact learning models. The primary distinction of this work from the most closely related work [41] is that the classes studied in this paper are not *determinate* because functions can be nested to arbitrary depth.[2]

6.1 The Model

Algorithms that learn concepts expressed in propositional logic traditionally have used as examples truth assignments, or models. Such an example is positive if and only if it satisfies the concept. But concepts in first-order logic may (and almost always do) have infinite models. Therefore, algorithms that learn definite clause[3] theories typically take logical formulas, usually ground atomic formulas, as examples instead. Such an example is positive if and only if it is a logical consequence of the concept. The algorithms in this chapter use ground atomic formulas (atoms) as examples in this manner. A concept is used to classify ground atoms according to the atoms' truth value in the concept's least Herbrand model,[4] which is

[2] More precisely, the classes that result from *flattening* (rewriting to contain no functions) the concepts are not determinate.

[3] A definite clause is a disjunction of atoms, exactly one of which is unnegated.

[4] The least Herbrand model of a set of definite clauses (every such set has one) is sometimes referred to as *the model* or the *unique minimal model* of the set. The set of definite clauses entails a logical sentence if and only if the sentence is true in this model. It is often useful to think of this model as a set, namely, the set of ground atoms that it makes true.

to say, according to whether the atoms logically follow from the concept. For example, the concept $[\forall x(p(f(g(x))))] \wedge [\forall x(p(f(x)) \rightarrow p(f(h(x))))]$, which is in $\mathcal{H}_{2,1}$, classifies $p(f(g(c)))$ and $p(f(h(h(h(g(c))))))$ as true or *positive* while it classifies $p(f(c))$ as false or *negative*. If A and B are two concepts that have the same least Herbrand model, we say they are *equivalent*, and we write $A \cong B$.

In a learning problem, a concept C, called the *target*, is chosen from some class of concepts $\mathcal{C}$ and is hidden from the learner. Each concept classifies each possible example element x from a set X, called the *instance space*, as either *positive* or *negative*. The learner infers some concept C' based on information about how the target C classifies the elements of the instance space X. For each of our learning problems, the concept class $\mathcal{C}$ is a class of definite clause theories, and we require that any learning algorithm, $\mathcal{A}$, must for any $C \in \mathcal{C}$ produce a concept $C' \in \mathcal{C}$ such that $C' \cong C$, that is, that C and C' have the same least Herbrand model. (For predictability we remove the requirement that C' belong to $\mathcal{C}$.) The instance space X is the Herbrand universe of C, and the learning algorithm $\mathcal{A}$ is able to obtain information about the way C classifies elements of X only by asking *equivalence queries*, in which $\mathcal{A}$ conjectures some C' and is told whether $C' \cong C$. If $C' \ncong C$, $\mathcal{A}$ is provided a *counterexample* x that C' and C classify differently.

We close this section by observing that the union of several classes can be learned by interleaving the learning algorithms for each class.

Fact 70 *Let $p(n)$ be a polynomial in n, and let $\{\mathcal{C}_i : 1 \leq i \leq p(n)\}$ be concept classes with learning algorithms $\{A_i : 1 \leq i \leq p(n)\}$ having time complexities $\{T_{A_i} : 1 \leq i \leq p(n)\}$ respectively. Then the concept class $\cup_{i=1}^{p(n)} \mathcal{C}_i$ can be learned in time $max_{1 \leq i \leq p(n)}\{p(n)T_{A_i}\}$.*

6.2 The Class $\mathcal{H}_{2,1}$

The concept class $\mathcal{H}_{2,1}$ is the class of concepts that can be expressed as a conjunction of at most two simple clauses, where a simple clause is a positive literal (an atom) composed of unary predicates and unary or 0-ary functions or an implication between two such positive literals. Allowing arbitrarily many literals in the antecedent of the implication, rather than only one, provides no additional expressivity. This is because we are considering the least Herbrand model to be the meaning of a concept in $\mathcal{H}_{2,1}$. The least Herbrand model of a pair of implicative clauses

is empty; on the other hand if one of the clauses is an atom A, the implicative clause will add atoms to the least Herbrand model only if *every* literal in the antecedent of the implicative clause unifies with A.

As an example, the following is a concept in $\mathcal{H}_{2,1}$ that we have seen already.

$$[\forall x(p(f(g(x))))] \wedge [\forall x(p(f(x)) \rightarrow p(f(h(x))))]$$

Since our conjuncts are always universally quantified, we henceforth leave the quantification implicit. Thus the above concept is written

$$[p(f(g(x)))] \wedge [p(f(x)) \rightarrow p(f(h(x)))]$$

We can divide $\mathcal{H}_{2,1}$ into two classes: trivial concepts, which are equivalent ($\cong$) to conjunctions of at most two atoms, and non-trivial, or recursive, concepts. The trivial concepts of $\mathcal{H}_{2,1}$ can be learned easily.[5] We next describe an algorithm that learns the non-trivial, or recursive, concepts in $\mathcal{H}_{2,1}$. It follows that $\mathcal{H}_{2,1}$ is learnable, since we can interleave this algorithm with the one that learns the trivial concepts of $\mathcal{H}_{2,1}$.

It can be shown that the recursive concepts in $\mathcal{H}_{2,1}$ have the form

$$[p(t_1)] \wedge [p(t_2(x)) \rightarrow p(t_3(x))] \tag{6.1}$$

where t_1 is a term, and $t_2(x)$ and $t_3(x)$ are terms ending in the same variable, x.[6] The fact that the functions and predicates are unary leads to a very concise description of a recursive concept in $\mathcal{H}_{2,1}$. Specifically, we can drop all parentheses in and around terms. Further, since we are discussing recursive concepts, all predicate symbols are the same and can likewise be dropped. Thus any concept having the form of concept (6.1) may be written $[\alpha e] \wedge [\beta x \rightarrow \gamma x]$, or

$$\begin{cases} \alpha e \\ \beta x \rightarrow \gamma x \end{cases} \tag{6.2}$$

[5] The basic idea is that the learning algorithm, by using equivalence queries, is able to obtain one example for each of the (at most two) atoms in the concept. Only a few atoms are more general than (that is, have as instances) each example, and the algorithm conjectures all combinations of these atoms, one for each of the (at most two) examples.

[6] The proof of this is omitted for brevity, as are some other proofs.

where α, β, and γ are strings of function symbols, x is a variable, and e is either a constant or a variable. Using this notation, determining whether, for example, αe unifies with βx requires only determining whether either α is a prefix of β or β is a prefix of α. For any strings α and β, if α is a prefix of β then we write $\alpha<\beta$.

The atoms in the least Herbrand model that are not instances of the *base atom* in the concept[7] are generated by applying the recursive clause. A concept is equivalent to a conjunction of two atoms if its recursive clause can be applied at most once. For the recursive clause to apply at all, α must unify with β, and for it to apply more than once, β must unify with γ. Hence a concept is non-trivial only if either $\alpha \leq \beta$ or $\beta \leq \alpha$ and either $\beta \leq \gamma$ or $\gamma \leq \beta$. In light of Fact70, to show that the non-trivial concepts can be learned in polynomial time, we need show only that the class of non-trivial concepts can be partitioned into a polynomial number of concept classes, each of which can be learned in polynomial time. Therefore, we carve the class of non-trivial concepts into five different subclasses defined by the prefix relationships among α, β, and γ. The approach for each subclass is similar—generalize the first positive example in such a way that the oracle is forced to provide a positive example containing whatever pieces of α, β, or γ are missing from the first example. The five possible sets of prefix relationships that can yield recursive concepts, based on our earlier discussion, are (1) $\beta<\alpha$ and $\beta<\gamma$, (2) $\alpha<\beta$ and $\beta<\gamma$, (3) $\alpha<\gamma$ and $\gamma<\beta$, (4) $\alpha<\beta$ and $\gamma<\alpha$, (5) $\beta<\alpha$ and $\gamma<\beta$. We do not need to divide case (1) into two cases because the relationship between α and γ is irrelevant here.

6.2.1 $\beta<\alpha$ and $\beta<\gamma$

This class consists of concepts with the form

$$\begin{cases} \phi\psi e \\ \phi x \rightarrow \phi\omega x \end{cases} \tag{6.3}$$

Concepts of this form insert arbitrarily many copies of ω between ϕ and ψ. Thus a concept of this class has for its least Herbrand model

$$\phi\omega^*\psi e$$

[7] Since we are speaking now of recursive concepts only, we refer to the two parts of the concept as the *base atom* and the *recursive clause*.

Lemma 71 *This class can be learned in polynomial time.*

Proof: To learn this class an algorithm needs to obtain an example that contains ϕ, ω, and ψ. Every example contains ϕ and ψ, and every example except an instance of the base atom contains ω. The algorithm first conjectures the null concept and receives a positive counterexample αe, which is an instance of some atom generated by the concept (either the base atom or the result of applying the recursive clause some number of times). There are $|\alpha| + 1$ atoms of which αe is an instance—αe itself and each $\alpha_i x$ where α_i is a prefix of α. The algorithm *guesses* one of these as the generated atom. (Here and throughout, when we use the term *guesses*, we actually mean that the algorithm dovetails all the possible choices.) It then proposes the generated atom in an equivalence query and receives another positive counterexample, which is an instance of some other generated atom. The algorithm guesses this generated atom from this example in a similar manner. Then the algorithm guesses which of the two generated atoms is generated later; this atom must contain ϕ, ω, ψ (and e if e is a constant). Finally, the algorithm guesses whether e is a constant or a variable,[8] and, from the later atom guesses the values of ϕ, ω, and ψ. It is straightforward to verify that there are at most $2n^3$ possible combinations of such guesses. □

6.2.2 $\alpha<\beta$ and $\beta<\gamma$

This class consists of concepts with the form

$$\begin{cases} \phi e \\ \phi\psi x \rightarrow \phi\psi\omega x \end{cases} \tag{6.4}$$

Concepts of this form add copies of ω after a prefix of $\phi\psi$. This class has as its least Herbrand model

$$\phi e + \phi\psi\omega^*$$

provided ϕe unifies with $\phi\psi x$, otherwise the least Herbrand model degenerates to ϕe.

Lemma 72 *This class is exactly learnable.*

[8] If e is a constant, it is the last symbol in each example.

Proof Sketch: Let ρe be the the (positive) counterexample to the empty hypothesis. Assuming that $\rho = \phi\psi\omega^k\zeta$ for some $k > 0$, $O(|\rho|^4)$ guesses suffice to identify ϕ, ψ, and ω.

If ρ does not contain a copy of ω, then $\rho = \phi\zeta$ and $O(|\rho|)$ guesses suffice to identify ϕ. Once ϕ has been identified, conjecturing

$$\left\{ \phi e \right. \tag{6.5}$$

will elicit a (positive) counterexample $\rho' e$ that does contain copies of both ψ and ω; identifying each of these requires only $O(|rho'|^3)$ guessès. □

6.2.3 $\alpha<\gamma$ and $\gamma<\beta$

This class consists of concepts with the form

$$\left\{ \begin{array}{l} \phi e \\ \phi\psi\omega x \rightarrow \phi\psi x \end{array} \right. \tag{6.6}$$

The recursive rule adds nothing to the least Herbrand model, so concepts of this form have as the least Herbrand model

$$\phi e$$

, so concepts of this form are equivalent to

$$\left\{ \phi e \right. \tag{6.7}$$

.

Lemma 73 *This class is exactly learnable.*

Proof Sketch: Let ρe be the the (positive) counterexample to the empty hypothesis. Because $\rho = \phi\zeta$, only $O(|\rho|)$ guesses are needed to identify ϕ. □

6.2.4 $\alpha<\beta$ and $\gamma<\alpha$

This class consists of concepts with the form

$$\begin{cases} \phi\psi e \\ \phi\psi\omega x \rightarrow \phi x \end{cases} \tag{6.8}$$

Again the recursive rule adds nothing to the least Herbrand model, so concepts of this form have as the least Herbrand model

$$\phi' e$$

, where $\phi' = \phi\psi$. Thus concepts of this form are equivalent to

$$\begin{cases} \phi' e \end{cases} \tag{6.9}$$

.

Lemma 74 *This class is exactly learnable.*

Proof Sketch: Let ρe be the the (positive) counterexample to the empty hypothesis. Because $\rho = \phi'\zeta$, only $O(|\rho|)$ guesses are needed to identify ϕ'. □

6.2.5 $\beta<\alpha$ and $\gamma<\beta$.

This class consists of concepts having the form

$$\begin{cases} \phi\psi\omega e \\ \phi\psi x \rightarrow \phi x \end{cases} \tag{6.10}$$

Concepts of this form generate smaller and smaller atoms by deleting copies of ψ at the front of ω; if ψ is not a prefix of ω, the concept can delete only one copy of ψ. Any concept of this form has the least Herbrand model described by

$$\phi\psi^k\zeta e \quad \text{for } 1 \leq k \leq n+1, \text{ where } \omega = \psi^n\zeta$$

Lemma 75 *This class can be learned in polynomial time.*

Proof: To learn this class an algorithm needs to obtain an example that contains ϕ, ψ, ω, and ζ. It then must determine n. We give an algorithm that makes at most two equivalence queries to obtain ϕ, ψ, ω, and ζ. It then guesses larger and larger values for n until it guesses the correct value. This value of n is linearly related to the length of the base atom, so overall the algorithm takes polynomial time.

1. Conjecture the false concept to obtain counterexample ρe
2. Dovetail the following algorithms
   - Assuming $\rho = \phi\psi^j\zeta\xi e$ for some $j \geq 1$
     (a) Select ϕ, ψ, ζ, ξ from ρ
     (b) Guess the value of n
     (c) Halt with output
     $$\begin{cases} \phi\psi^{n+1}\zeta e \\ \phi\psi x \rightarrow \phi x \end{cases}$$
   - Assuming $\rho = \phi\zeta\xi e$
     (a) Select ϕ, ζ, ξ from ρ
     (b) Conjecture
     $$\{\ \phi\zeta e$$
     to obtain counterexample $\rho' e'$. Note that ρ' necessarily contains ψ as a substring.
     (c) Select ψ from ρ'
     (d) Guess n
     (e) Halt with output
     $$\begin{cases} \phi\psi^{n+1}\zeta e \\ \phi\psi x \rightarrow \phi x \end{cases}$$

When the algorithm *selects* substrings from a counterexample, it is in reality dovetailing all possible choices; nevertheless, we observe that there are only $O(|\rho|^5)$ (respectively $O(|\rho'|)$) choices to try. Similarly, when the algorithm guesses the value of n, it is actually making successively larger and larger guesses for n and testing whether it is correct with an equivalence query. It will obtain the correct value for n in time polynomial in the size of the target concept. At that point it outputs the necessarily correct concept and halts. □

Some observations about the above example are in order. In these, and afterward, we use g_0 to denote the base atom and, inductively, g_{i+1} to denote the result of applying the recursive

clause to g_i.[9] It is worth noting that the algorithm above uses only two of the counterexamples it receives, though it typically makes more than two equivalence queries. This is the case with the algorithms for the other subclasses as well. It is also worth noting that when the algorithm above guesses the value of n, it is guessing the number of times the recursive clause is applied to generate the earliest generated atom g_i of which either example is an instance.

From the preceding arguments, we have Theorem 76, below.

Theorem 76 *The concept class $\mathcal{H}_{2,1}$ is learnable.*

6.3 Increasing Predicate Arity

It is often useful to have predicates of higher arity, but otherwise maintain the form of the concepts in $\mathcal{H}_{2,1}$. For example

$$\begin{cases} plus(x, 0, x) \\ plus(x, y, z) \rightarrow plus(x, s(y), s(z)) \end{cases} \qquad \begin{cases} greater(s(x), 0) \\ greater(x, y) \rightarrow greater(s(x), s(y)) \end{cases}$$

In this section we remove the requirement that predicates be unary. Specifically, let $\mathcal{H}_{2,*}$ be the result of allowing predicates of arbitrary arity but requiring functions to remain unary, with the additional restriction, which we define next, that variables be *stationary*. Notice that because functions are unary, each argument has at most one variable (it may have a constant instead of a variable), and that variable must be the last symbol of the argument. A concept meets the *stationary variables* restriction if for any variable x, if x appears in argument i of the consequent of the recursive clause then x also appears in argument i of the antecedent. This class does include the above arithmetic concepts built with the successor function and the constant 0, but does not include the concept $[p(a, b, c)] \wedge [p(x, y, z) \rightarrow p(z, x, y)]$ because variables "shift positions" in the recursive clause.

We begin with the following observation. As for the class $\mathcal{H}_{2,1}$, a concept in $\mathcal{H}_{2,*}$ is trivial if it is equivalent to a conjunction of at most two atoms. It is straightforward to verify that, because variables are stationary, the non–trivial concepts in $\mathcal{H}_{2,*}$ have the form

$$[p(t_1(e_1), ..., t_k(e_k))] \wedge [p(t'_1(e'_1), ..., t'_k(e'_k)) \rightarrow p(t''_1(e''_1), ..., t''_k(e''_k))] \qquad (6.11)$$

[9] If the concept is a conjunction of two atoms, the choice of which atom is g_0 and which is g_1 is arbitrary.

where $t_i(e_i)$, $t'_i(e'_i)$, and $t''_i(e''_i)$ denote terms ending in e_i, e'_i, and e''_i, respectively, and for some $1 \leq j \leq k$ we have $e'_j = e''_j = x$ for some variable x.

Unfortunately, we have the following result for the class $\mathcal{H}_{2,*}$.

Theorem 77 *$\mathcal{H}_{2,*}$ is not learnable, assuming $RP \neq NP$.*[10]

Before proving the theorem, we prove the following useful lemma.

Lemma 78 *Any concept in $\mathcal{H}_{2,*}$ generates at most two function-free atoms, that is, at most two atoms whose arguments are constants or variables.*

Proof: The result clearly holds for the trivial concepts in $\mathcal{H}_{2,*}$. Any other concept has form 6.11 above. Suppose a concept of this form generates at least three function-free atoms. One of these may be the base atom itself, while at least two of these atoms must be generated using the recursive clause. For the recursive clause to be used in generating such atoms, its consequent $p(t''_1(e''_1), ..., t''_k(e''_k)))$ must be function-free, that is, must be of the form $p(e''_1, ..., e''_k))$, where each e''_i is a constant or a variable. Let $p(d_1, ..., d_k)$, where each d_i is a constant or a variable, be the first function-free atom generated using the recursive clause. We show that any atom generated after $p(d_1, ..., d_k)$ is also $p(d_1, ..., d_k)$; hence, no third function-free atom is generated by the concept.

For the recursive clause to generate an atom after $p(d_1, ..., d_k)$, $p(d_1, ..., d_k)$ must unify with the antecedent of the recursive clause, $p(t'_1(e'_1), ..., t'_k(e'_k))$. Therefore, we know that for each $1 \leq i \leq k$, at least one of the following is true: $d_i = t'_i(e'_i) = c$ for some constant c, or $t'_i(e'_i)$ is a variable x, or d_i is a variable. In the first case, $e''_i = c$ (and in fact $e_i = c$), so every atom generated by the concept has $c = d_i$ at argument i. In the second case, if $e''_i = x$, then $e_i = d_i$, so every atom generated by the concept has d_i at argument i; otherwise, $e''_i = d_i$, so every atom generated by applying the recursive clause has d_i at argument i. In the third case, e''_i must be a variable y. We have just argued the case where $t'_i(e'_i)$ and e''_i are the same variable, x. If $t'_i(e'_i) \neq e''_i$, then $d_i = e''_i = y$, so every atom generated by applying the recursive clause has d_i at argument i. Thus every atom generated after $p(d_1, ..., d_k)$ agrees with $p(d_1, ...d_k)$ at every argument. □

[10] That is, $\mathcal{H}_{2,*}$ is not PAC learnable, and therefore is not learnable in polynomial time by equivalence queries. RP is the class of problems that can be solved in random polynomial time.

Proof of Theorem 77: We show that the *consistency problem* [85] for this concept class is NP-hard. Our reduction is from the consistency problem for two-term-DNF. The consistency problem is: given a set of labeled examples, determine whether there exists a concept in the class that agrees with all examples on their labels. In particular, the consistency problem for $\mathcal{H}_{2,\bullet}$ requires determining whether there exists a concept in $\mathcal{H}_{2,\bullet}$ that entails all of the positive examples but none of the negative ones. If the consistency problem for a given class is NP-hard, then that the class is not PAC learnable (assuming $RP \neq NP$) [85], which in turn implies that it is not polynomial-time learnable by equivalence queries.

We first claim, based on Lemma 78, that if all examples are function–free then $\mathcal{H}_{2,\bullet}$ contains a concept consistent with the examples if and only if there exists a function–free conjunction of at most two atoms[11] that is consistent with the examples. To see the *if* case, notice that every function–free conjunction of at most two atoms is in $\mathcal{H}_{2,\bullet}$. To see the *only if* case, notice that only function–free atoms can have other function-free atoms as instances; therefore, a function–free atom is classified positive by a given concept in $\mathcal{H}_{2,\bullet}$ if and only if it is an instance of one of the (at most two, by Lemma 78) function–free atoms generated by the concept.

Therefore, it is enough to show that the problem of finding a function–free conjunction of at most two atoms consistent with a labeled sample of function–free atoms is NP–hard. (Doing so will in fact show that the consistency problem is NP-complete, since we can efficiently check whether a given conjunction of at most two atoms is consistent with a labeled sample.) To do so, we reduce from the consistency problem for 2-term-DNF [85].

Define a mapping $f : \{0,1\}^n \rightarrow \{0,1,z_1,z_2,\ldots,z_n\}^n$ as follows: $y = f(x)$ where $y^i = 1$ and $y^{n+i} = z_i$ when $x^i = 1$, and $y^i = z_i$ and $y^{n+i} = 0$ when $x^i = 0$.

Now consider any labeled sample S and construct $S' = \{p(y)|x \in S, y = f(x)\}$ where S' preserves the labels of S. We show that there is a two atom conjunction consistent with S' if and only if there is a 2-term-DNF consistent with S.

If: Define a mapping g from (propositional) terms, t, to atoms as follows. Start with the atom $p(z_1,z_2,\ldots,z_n)$. If the literal x_i appears in t, change z_i to 1. If the literal $\bar{x}_i$ appears in t, change z_{n+i} to 0. Notice that for any (propositional) term t and any example x, every instance

[11] That is, a single function–free atom or a conjunction of two function–free atoms.

of $p(f(x))$ is an instance of $g(t)$ if and only if x satisfies t. Thus, $g(t_1) \wedge g(t_2)$ is consistent with S' if $t_1 \vee t_2$ is consistent with S.

Only If: Now consider a conjunction of two atoms consistent with S'. We must show how to construct a 2-term-DNF consistent with S.

Consider any two atom conjunction, $a_1 \wedge a_2$, consistent with S'. For each atom, a, construct a (propositional) term as follows. If argument $i (i \leq n)$ of a is 1, place x_i in the term. If argument $n+i$ of a is 0, place $\bar{x_i}$ in the term. If arguments i and j $(i < j \leq n)$ are the same, place both x_i and x_j in the term. If arguments $n+i$ and $n+j$ are the same, place both $\bar{x_i}$ and $\bar{x_j}$ in the term. We claim that the disjunction of the two terms so formed from a_1 and a_2 is a 2-term-DNF consistent with S.

Observe that we can form two (not necessarily disjoint) subsets, S'_1 and S'_2 of S' such that each atom in S'_1 is entailed by a_1 and each atom in S'_2 is entailed by a_2. Notice that neither S'_1 nor S'_2 contains any of the negative examples of S', but between them they contain all the positive examples of S'. Without loss of generality consider S'_1 and a_1. If the ith argument of a_1 is 1, then the ith argument of every atom in S'_1 is 1. But the atoms in S'_1 have 1 as their ith argument only if their (propositional) counterparts had x_i set to 1. Therefore, placing x_i in the (propositional) term arising from a_1 does not mis-label any of the (propositional) examples that gave rise to atoms in S'_1. Similarly if the $n+i$th argument of a_1 is 0.

Finally, notice that no atom in S' contains a co-reference (ie, no variable appears twice in an atom). Thus if a_1 contains a co-reference, the co-reference must correspond to arguments that were equal constants in the atoms in S'_1. However, any constants appearing among the first n arguments of an atom in S'_1 must be 1, while any constants appearing among the last n arguments of an atom in S'_1 must be 0. Therefore, we can make a_1 more specific by changing a co-reference among the first n arguments to a 1, and by changing a co-reference among the last n arguments to a 0; the resulting atom entails no more than a_1 but still entails every atom in S'_1.

A similar construction can be done for a_2 and S'_2. Disjoining the (propositional) terms so constructed produces a 2-term-DNF consistent with S. □

Our conjecture is that the class $\mathcal{H}_{2,*}$ is not even predictable (that is, learnable *in terms of* any other class), though this is an open question. Nevertheless, we now show that if we

fix the predicate arity to any integer k, then the resulting concept class $\mathcal{H}_{2,k}$ is learnable in terms of a slightly more general class, called $\mathcal{H}_{2,k}'$, and is therefore predictable (the question of learnability of $\mathcal{H}_{2,k}$ in terms of $\mathcal{H}_{2,k}$ itself remains open). Concepts in $\mathcal{H}_{2,k}'$ may be any union of a concept in $\mathcal{H}_{2,k}$ and two additional atoms built from variables, constants, unary functions, and predicates with arity at most k; an example is classified as positive by such a concept if and only if it is classified as positive by the concept in $\mathcal{H}_{2,k}$ or is an instance of one of the additional atoms. For the class $\mathcal{H}_{2,k}'$, we consider conjunctions of at most three atoms to be trivial concepts. The class of such concepts can be learned by a simple algorithm that is similar to the algorithm that learns the trivial concepts of $\mathcal{H}_{2,1}$. We now focus, instead, on the learning algorithm for the recursive concepts of $\mathcal{H}_{2,k}'$. This learning algorithm is based on the learning algorithm for $\mathcal{H}_{2,1}$, and central to it are the following definition and lemma. In the lemma, and afterward, we use G_0 to denote the base atom and, inductively, G_{i+1} to denote the result of applying the recursive clause to G_i. The following lemma allows us to learn $\mathcal{H}_{2,k}$ using the algorithm for learning $\mathcal{H}_{2,1}$.

Definition 79 *Let*

$$\begin{cases} p(\alpha_1 e_1, ..., \alpha_k e_k) \\ p(\beta_1 e_1', ..., \beta_k e_k') \rightarrow p(\gamma_1 e_1'', ..., \gamma_k e_k'') \end{cases}$$

be a concept in $\mathcal{H}_{2,k}$. Then we say the subconcept at argument i, *for $1 \leq i \leq k$, of this concept is*

$$\begin{cases} \alpha_i e_i \\ \beta_i e_i' \rightarrow \gamma_i e_i'' \end{cases}$$

Lemma 80 *Let*

$$\begin{cases} p(\alpha_1 e_1, ..., \alpha_k e_k) \\ p(\beta_1 e_1', ..., \beta_k e_k') \rightarrow p(\gamma_1 e_1'', ..., \gamma_k e_k'') \end{cases}$$

be a concept in $\mathcal{H}_{2,k}$. For any $1 \leq j \leq k$, if e_j' is a variable x, then for any $n \geq 2$: if $G_n = p(t_1, ..., t_k)$ unifies[12] with $p(\beta_1 e_1', ..., \beta_k e_k')$, the binding generated for x by this unification is the same as the binding generated for x by unifying t_j with $\beta_j x$.[13]

[12] Note that G_3 exists if and only if G_2 unifies with $p(\beta_1 e_1', ..., \beta_k e_k')$. If G_3 does not exist, the result holds trivially.

[13] The proof of this lemma is rather long. We present it here because the lemma is both non-obvious and central to one main result of the chapter, Theorem 81.

Proof: We prove the contrapositive. Assume that for some $1 \leq j \leq k$, e'_j is a variable x, and for some $n \geq 2$, $G_n = p(t_1, ..., t_k)$ unifies with $p(\beta_1 e'_1, ..., \beta_k e'_k)$, but the binding generated for x by this unification is not the same as the binding generated for x by unifying t_j with β_j. Then for some $1 \leq i \leq k$, e_i is also x, and unifying t_i with $\beta_i x$ yields a different binding for x than does unifying t_j with $\beta_j x$. We obtain a contradiction by showing that these unifications cannot yield different bindings.

Because the concept is *stationary*, and e'_1 and e'_j are each x, each of e''_i and e''_j must be either x or a constant. Furthermore, for the recursive clause to be applicable to generate G_3, if e''_i and e''_j are both constants then they must be some same constant, c. This leaves us three cases to consider.

First, both e''_i and e''_j are some constant c. Then the recursive rules for argument i and argument j, respectively, are $\beta_i x \rightarrow \gamma_i c$ and $\beta_j x \rightarrow \gamma_j c$. Hence G_1 has $\gamma_i c$ and $\gamma_j c$ at arguments i and j, respectively. Then since G_2 is generated, $\beta_i t = \gamma_i c$ and $\beta_j t = \gamma_j c$ for some term t. Then argument i and argument j always, individually bind x to t after G_2, giving a contradiction.

Second, e''_i is a constant, c, and e''_j is a variable, x. (By symmetry, this case also covers the case where e''_i is x and e''_j is c.) Then the recursive rules for argument i and argument j, respectively, are $\beta_i x \rightarrow \gamma_i c$ and $\beta_j x \rightarrow \gamma_j x$. Hence G_1 has $\gamma_i c$ at argument i. Since G_2 is generated, we have $\beta_i t = \gamma_i c$ for some ground term t. Then on unification of G_1 with $p(\beta_1 e'_1, ..., \beta_k e'_k)$, x is bound to t. Furthermore, it must be the case that either $\beta_j \leq \gamma_j$ or $\gamma_j \leq \beta_j$ for unification to succeed. Consider both cases.

Case 1: $\beta_j \leq \gamma_j$, so $\beta_j \phi = \gamma_j$ for some ϕ such that $|\phi| \geq 0$. Then G_2 has $\gamma_j t = \beta_j \phi t$ at argument j, and $\gamma_i c = \beta_i t$ at argument i. Since G_3 is generated, $\langle \beta_i t, \beta_j \phi t \rangle$ must unify with $\langle \beta_i x, \beta_j x \rangle$, which can occur only if $|\phi| = 0$. If $|\phi| = 0$, then after G_2 the recursive rules for argument i and argument j will always, independently bind x to t, giving a contradiction.

Case 2: $\gamma_j \leq \beta_j$, so $\gamma_j \phi = \beta_j$ for some ϕ such that $|\phi| \geq 0$. Then G_2 has $\gamma_j t$ at argument j and $\gamma_i c = \beta_i t$ at argument i. Since G_3 is generated, $\langle \beta_i t, \gamma_j t \rangle$ must unify with $\langle \beta_i x, \beta_j x \rangle = \langle \beta_i x, \gamma_j \phi x \rangle$, which can occur only if $|\phi| = 0$. If $|\phi| = 0$ then, again, after G_2 the recursive rules for argument i and argument j will always, independently bind x to t, giving a contradiction.

Finally, both e''_i and e''_j are the variable x. Then the recursive rules for argument i and argument j, respectively, are $\beta_i x \rightarrow \gamma_i x$ and $\beta_j x \rightarrow \gamma_j x$. Then G_2 has $\gamma_i t$ and $\gamma_j t$, for some

BEST COPY AVAILABLE

term t, at argument i and argument j, respectively. Since G_3 is generated, we have either $\beta_i \leq \gamma_i$ or $\gamma_i \leq \beta_i$. Consider both cases.

Case 1: $\beta_i \leq \gamma_i$, so $\beta_i\phi = \gamma_i$ for some ϕ such that $|\phi| \geq 0$. For G_2 to be generated, $\langle \gamma_i t, \gamma_j t \rangle$ must unify with $\langle \beta_i x, \beta_j x \rangle$. This is possible only if $\beta_j\phi = \gamma_j$, in which case, during unification with G_m, $m \geq 1$, the recursive rules for argument i and argument j will always, independently bind x to $\phi^m t$, giving a contradiction.

Case 2: $\gamma_i \leq \beta_i$, so $\beta_i = \gamma_i\phi$ for some ϕ such that $|\phi| \geq 0$. For G_2 to be generated, $\langle \gamma_i t, \gamma_j t \rangle$ must unify with $\langle \beta_i x, \beta_j x \rangle = \langle \gamma_i\phi x, \beta_j x \rangle$. For this to occur, either (A) $t = \psi y$, for some variable y, and $\psi \leq \phi$, or (B) $t = \psi e$, for some constant or variable e, and $\phi \leq \psi$. If (A) is true, then since $t = \psi y$ where $\psi \leq \phi$, we have $\psi\omega = \phi$ for some ω such that $|\omega| \geq 0$. Since $\langle \gamma_i t, \gamma_j t \rangle = \langle \gamma_i\psi y, \gamma_j\psi y \rangle$ must unify with $\langle \gamma_i\phi x, \beta_j x \rangle = \langle \gamma_i\psi\omega x, \beta_j x \rangle$, it must be the case that $\beta_j = \gamma_j\psi\omega$. Then x is left unbound during unification, so G_3 has $\gamma_i x$ and $\gamma_j x$ at arguments i and j, respectively. Then for each G_m, $m \geq 3$, x remains unbound, and G_m has $\gamma_i x$ and $\gamma_j x$ at arguments i and j respectively. Because x remains unbound throughout, bindings given by the recursive rules for arguments i and j cannot disagree—a contradiction. If (B) is true, then since $\phi \leq \psi$, we have $\psi = \phi\omega$ for some ω such that $|\omega| \geq 0$. For $\langle \gamma_i t, \gamma_j t \rangle = \langle \gamma_i\psi e, \gamma_j\psi e \rangle = \langle \gamma_i\phi\omega e, \gamma_j\phi\omega e \rangle$ to unify with $\langle \gamma_i\phi x, \beta_j x \rangle$, it must be the case that $\beta_j = \gamma_j\phi$. Then x is bound to ωe. The recursive rules for arguments i and j therefore have the forms, respectively, $\gamma_i\phi x \rightarrow \gamma_i x$ and $\gamma_j\phi x \rightarrow \gamma_j x$. Hence G_m, $m \geq 3$, is generated if and only if $\phi\omega = \phi^n\eta$ for some η, where $n \geq (m-2)$. In unifying G_m, $m \geq 2$, with $p(\beta_1 e_1, ..., \beta_k e_k)$, the recursive rules for arguments i and j will always, independently bind x to $\phi^{n-m}\eta$—a contradiction. □

Theorem 81 *For any constant k, the class $\mathcal{H}_{2,k}$ is predictable from equivalence queries alone.*

Proof: (Sketch) By Lemma 80, any concept C in $\mathcal{H}_{2,k}$ can be rewritten as the union of three concepts (only polynomially larger than C), two of which are the atoms G_0 and G_1. The third concept is some $\tilde{C}$ in $\mathcal{H}_{2,k}$ whose base atom $\tilde{G}_0$ is G_2 and whose recursive clause (if any) generates $\tilde{G}_1 = G_3, \ldots, \tilde{G}_m = G_{m+2}, \ldots$ meeting the following condition: for all $m \geq 0$ and any variable x in the antecedent of the recursive clause, no two argument positions impose different bindings on x when generating G_{m+1} from G_m. In other words, the behavior of $\tilde{C}$ can be understood as a simple composition of the behavior of the subconcepts at arguments 1 through k. This observation motivates an algorithm that learns $\mathcal{H}_{2,k}$ in terms of $\mathcal{H}_{2,k}'$. At the highest

level of description, the algorithm poses equivalence queries in such a way that it obtains, as examples, instances of G_0, G_1, and $\tilde{G}_i$ and $\tilde{G}_j$ for distinct i and j. The algorithm determines G_0 and G_1 from their examples, and it determines $\tilde{C}$ from $\tilde{G}_i$ and $\tilde{G}_j$. To determine $\tilde{C}$ the algorithm uses the learning algorithm for $\mathcal{H}_{2,1}$ to learn the subconcepts of $\tilde{C}$.[14] We now fill in the details of the algorithm.

The algorithm begins by conjecturing the empty theory, and it necessarily receives a positive counterexample in response. This counterexample is an instance of some more general atom, A_1, that is either G_0, G_1, or some $\tilde{G}_i$. The algorithm guesses A_1 and guesses whether A_1 is G_0, G_1, or some $\tilde{G}_i$. It then conjectures A_1 in an equivalence query and, if A_1 is not the target, necessarily receives another positive example. (As earlier, by *guess* we mean that the algorithm dovetails the possible choices.) This example is also an instance of G_0, G_1, or some $\tilde{G}_i$, but it is *not* an instance of A_1. Again, the algorithm guesses that atom—call it A_2—and guesses whether it is of G_0, G_1, or some $\tilde{G}_i$. Following the second example, and following each new example thereafter, the algorithm has at least two of of G_0, G_1, $\tilde{G}_i$, and $\tilde{G}_j$ (some $i \neq j$). It conjectures the union of those that it has, with the following exception: if it has both $\tilde{G}_i$, and $\tilde{G}_j$ (any $i \neq j$), it uses a guess of $\tilde{C}$ in place of $\tilde{G}_i$ and $\tilde{G}_j$. Again, in response to such a conjecture, either the algorithm is correct or it receives a positive counterexample. It remains only to show how (1) the atoms G_0, G_1, $\tilde{G}_i$, and $\tilde{G}_j$ are "efficiently guessed", and (2) $\tilde{C}$ is "efficiently guessed" from $\tilde{G}_i$ and $\tilde{G}_j$.

Given *any* atom, A, entailed by G_0, we know that G_0 is a generalization A. Because no function has arity greater than 1, there are at most $O((2|A|)^k)$ generalizations of A to consider, all of which can be tried in parallel. Note that, because k is fixed, the number of possible generalizations is polynomial in the size of the example. G_1, $\tilde{G}_i$, $\tilde{G}_j$ are efficiently guessed in the same way.

To learn $\tilde{C}$, the algorithm first determines the *high-level structure* of $\tilde{C}$; specifically, it guesses how many variables are in the base atom and in the antecedent and consequent of the recursive

[14] The only subtlety is that the learning algorithm for $\mathcal{H}_{2,1}$ might return a concept that is a conjunction of at most two atoms; such a concept cannot serve as a subconcept for a member of $\mathcal{H}_{2,k}$. But the only case in which this can occur is the case for which every atom after g_0 is the same. And we provide the learning algorithm with only examples of g_2, g_3, ... (examples extracted from G_2, G_3, ...). Therefore, if the concept returned by the learning algorithm is a conjunction of at most two atoms, it is in fact a single atom, αe, in which case we may use the concept in $\mathcal{H}_{2,1}$ whose base atom, recursive clause antecedent, and recursive clause consequent are all αe.

clause of $\tilde{C}$ and which subconcept uses which (if any) variable. There are at most $O(k^{3k})$ possibilities, where k is fixed. The algorithm is then left with the task of precisely determining the subconcepts of $\tilde{C}$. Because it has two examples of distinct (most general) atoms generated by $\tilde{C}$—$\tilde{G}_i$ and $\tilde{G}_j$—it has two examples of each subconcept. The algorithm uses the learning algorithm for $\mathcal{H}_{2,1}$ to learn these subconcepts from these examples, with the following slight modification. While the learning algorithm for $\mathcal{H}_{2,1}$ would ordinarily conjecture a concept in $\mathcal{H}_{2,1}$, the present learning algorithm must conjecture a concept in $\mathcal{H}_{2,k}{}'$. Therefore, the algorithm conjectures every concept in $\mathcal{H}_{2,k}{}'$ that results from any combination of conjectures, by the learning algorithm for $\mathcal{H}_{2,1}$, for the subconcepts; that is, it tries all combinations of subconcepts. Because k is fixed, the number of such combinations is polynomial in the sizes of the counterexamples seen thus far. Finally, recall that in some cases (as in concept 6.10, page 4) the learning algorithm for $\mathcal{H}_{2,1}$ must guess the value for n, where n is the smallest number of times the recursive clause is applied to generate one of the examples.[15] Therefore, the present learning algorithm may have to guess n. This is handled by initially guessing $n = 1$ and guessing successively higher values of n until the correct n is reached. This approach succeeds provided that the target truly contains a type 6.10 subconcept. In the case that the target does not contain a type 6.10 concept, the potentially non-terminating, errant search for a non-existent n halts because we are interleaving the steps form *all* pending guesses of *all* forms—including the correct non-type 6.10 form—of the subconcept. □

6.4 Increasing Function Arity

This section provides some evidence that the techniques introduced in the previous sections are useful in studying the learnability of other classes of definite clause theories with recursion. In this particular, this section considers the concept class $TP^2 \cup CFTFB_{uniq}$ studied by Arimura, Ishizaka, and Shinohara [15]. They show that $TP^2 \cup CFTFB_{uniq}$ is *learnable in the limit* with polynomial update time, from positive examples only. Until now, the learnability of this class in the *PAC learning* and *equivalence query* models has been unknown. Using techniques that allowed us to show that the class $\mathcal{H}_{2,*}$ is not learnable, this section shows that the class $TP^2 \cup CFTFB_{uniq}$ is *not* learnable (in both models), assuming $P \neq RP$.

[15] The n sought is necessarily the same for all subconcepts; thus only on n needs to be found.

Arimura, Ishizaka, and Shinohara define the class $CFTFB$ to contain each *Prolog program P defining the predicate p with at most two clauses C_0 and C_1*

$$C_0 = p(s_1, ..., s_k)$$

$$C_1 = p(x_1, ..., x_k) \rightarrow p(t_1, ..., t_k)$$

meeting the following additional conditions.

1. Every argument s_i $(1 \leq i \leq k)$ of the head of C_0 is either a function symbol of arity 0 or a variable symbol.
2. All arguments $x_1, ..., x_k$ of the body of C_1 are mutually distinct variables.
3. For every $1 \leq i \leq k$, every argument x_i of the body of C_1 occurs exactly once in the term t_i of the head. Moreover, x_i does not occur in any arguments t_j $(i \neq j)$ of the head.

In addition, the predicate arity k is fixed (as in the class $\mathcal{H}_{2,k}$, for example). Notice that another way to state condition (1) is that C_0 is function-free. The class $CFTFB_{uniq}$ is the subclass of $CFTFB$ whose concepts have a unique *2-mmg*; a 2-mmg is a pair of maximally-specific atoms whose instances include all the ground atoms in the concept's least Herbrand model (the ground atoms that logically follow from the concept). The class TP^2 is the class of all conjunctions of atoms (again the bound on predicate arity is assumed, though, again, no bound is placed on function arity). The class $TP^2 \cup CFTFB_{uniq}$ obviously contains exactly the concepts in TP^2 or $CFTFB_{uniq}$.

Theorem 82 *The class $TP^2 \cup CFTFB_{uniq}$ is not learnable, assuming $P \neq RP$.*

Proof: [Sketch] The proof is a simple variant of the proof of Theorem 77. Instead of function-free examples built from an n-ary predicate p, where n is arbitrary, we use examples built from a unary predicate p (thus the proof works regardless of how small k is restricted to be) and an n-ary function f. The proof is modified only slightly to replace claims about function-free concepts by analogous claims about concepts of the form $p(f(t_1, ..., t_n))$, where $t_1, ..., t_n$ are either constants or variables (are function-free). □

We now define the class $CFTFB'$ to be the union of $CFTFB$ and conjunctions of at most two atoms, one of which is function-free. In a way, this class is a more natural extension of $CFTFB$, in that it can also be described as the union of a function-free atom C_0 and a clause C_1 which may either be an atom or a clause of the form $p(x_1, ..., x_m) \rightarrow p(t_1, ..., t_m)$ meeting the restrictions given above. Notice also that this class is in some ways broader than $TP^2 \cup CFTFB_{uniq}$, since it includes those concepts in $CFTFB$ that do not have a unique 2-mmg. We conjecture that $CFTFB'$ can be learned using methods similar to those used in this chapter for $\mathcal{H}_{2,1}$ and $\mathcal{H}_{2,k}$. This is an important area for further work since (1) it would be the first positive learnability result, to our knowledge, for a non-determinate class of recursive concepts with functions of arity greater than one, and (2) it would provide additional evidence for the utility of the techniques presented in this chapter.

6.5 Increasing the Number of Clauses: The Class $\mathcal{H}_{*,1}$

In this section we return to the class $\mathcal{H}_{2,1}$ and extend it along a different dimension, allowing an arbitrary number of clauses. In other words, we consider concepts that consist of arbitrarily many definite clauses with antecedents of at most one literal where again all literals in the clauses are built from unary predicates and from constants, variables, and unary functions. We call this class $\mathcal{H}_{*,1}$. The following example is a concept in $\mathcal{H}_{*,1}$ that defines the even integers using the constant 0 and unary functions s for successor and p for predecessor (of course, under this definition each integer is represented by infinitely many different terms).

$even(0)$
$even(x) \rightarrow even(s(s(x)))$
$even(x) \rightarrow even(s(p(x)))$
$even(x) \rightarrow even(p(s(x)))$
$even(x) \rightarrow even(p(p(x)))$

As in the previous section, because all literals have the same predicate and all functions are unary, we may disregard the predicate and view the literals as strings. And again, we may assume that for any recursive clause, both literals end with some same variable x. In this section we introduce *prefix grammars* and *prefix languages*, and we indicate how the concepts in $\mathcal{H}_{*,1}$ may be viewed as prefix grammars. Moreover, every prefix grammar is equivalent to

a concept in $\mathcal{H}_{*,1}$, where, by equivalent, we mean that the set of strings generated by the grammar is exactly the least Herbrand model of the concept, when the members of this model are viewed as strings. The major result of this section is that the class of languages generated by prefix grammars, and therefore the class of least Herbrand models of concepts in $\mathcal{H}_{*,1}$, is the same as the class of regular languages. While this result alone does not lead to a polynomial-time learning algorithm, it does lead to a weaker learning algorithm, and it also shows that the answers to open questions about prefix grammars will provide learnability results (either positive or negative) for $\mathcal{H}_{*,1}$.

Definition 83 (Prefix Grammar) *A prefix grammar is a triple*

$$G = (\Sigma, S, P)$$

where Σ is a finite set of symbols, S is a finite set of strings over Σ, and P is a finite set of productions. Each production in P has the form $\alpha \rightarrow \beta$ where α and β are strings over Σ. The sentential forms for G are the strings over Σ alone.[16] *A production $\alpha \rightarrow \beta$ may be applied to a string γ if and only if $\gamma = \alpha\omega$ for some ω; the result of applying the production to γ is the string $\beta\omega$. $L(G)$ contains exactly the strings ω such that either: $\omega \in S$ or ω is the result of applying a production in P to a string in $L(G)$.*

Prefix grammars are so named because their productions may be applied only to a prefix of a string, to rewrite that prefix. Note that, in contrast with ordinary grammars, every sentential form generated by a prefix grammar G is a member of $L(G)$. The prefix languages are exactly the languages generated by prefix grammars.

The following example shows the set of definite clauses for the predicate *even* from the previous example rewritten as a prefix grammar. Predicate symbols are omitted, literals are treated as strings, and variables in the recursive clauses are treated as the empty string. The grammar generates strings in precisely the same way that the set of definite clauses generates atoms. The symbol ϵ denotes the empty string.

$\Sigma = \{0, s, p\}$

$S = \{0\}$

[16] That is to say, there are no non-terminal symbols in this grammar.

$$P = \{\epsilon \rightarrow ss, \epsilon \rightarrow sp, \epsilon \rightarrow ps, \epsilon \rightarrow pp\}$$

Theorem 84 *The classes of prefix languages and regular languages are the same. Specifically, for any prefix grammar there exists a right-linear grammar only polynomially larger that generates the same language. And for any DFA, there exists a prefix grammar that generates the language accepted by the DFA, and the grammar is at most exponentially larger than the DFA.*

Because the proof is relatively long, we spread it over the following two sections of the paper.

6.6 The Regular Languages Contain the Prefix Languages

We present an algorithm that transforms a prefix grammar G_P into a right-linear grammar G_R only polynomially larger than G_P. We assume that G_P has no productions of the form $\epsilon \rightarrow \delta$, because we can eliminate all such productions with at most polynomial growth of G_P as follows. We assume δ is nonempty, since any production $\epsilon \rightarrow \epsilon$ can be deleted. For each production $\epsilon \rightarrow \delta$ and every symbol c in G_P, add the production $c \rightarrow \delta c$ to G_P, and if ϵ is a production of G_P, add δ to the base set S of G_P. Remove every production $\epsilon \rightarrow \delta$ from G_P.

6.6.1 Prefix Grammar to Right-Linear Grammar Transformation Algorithm

Input: A prefix grammar G_P having base set $\{\alpha_1, ..., \alpha_n\}$ and productions $\{\beta_1 \rightarrow \gamma_1, ..., \beta_m \rightarrow \gamma_m\}$.

Output: A right-linear grammar G_R for $L(G_P)$.

We define a grammar G_R with start symbol S, nonterminals $V_{\beta_1}, ..., V_{\beta_m}$, and all symbols of G_P as terminals. Let G_R contain the following productions.

$$S \rightarrow \alpha_1 \mid ... \mid \alpha_n \mid \gamma_1 V_{\beta_1} \mid ... \mid \gamma_m V_{\beta_m}$$

In addition, add to G_R productions of the following form until no more productions can be added.

$V_{\beta_i} \rightarrow \omega V_{\beta_j}$ where $\quad S \rightarrow \phi_1 V_{\beta_{i_1}}$

$V_{\beta_{i_1}} \rightarrow \phi_2 V_{\beta_{i_2}}$

$$V_{\beta_{i_{k-1}}} \rightarrow \phi_k \omega V_{\beta_j}$$

are productions already in G_R, $\phi_1, ..., \phi_k$ are strings of terminals of G_R, $\phi_1\phi_2...\phi_k = \beta_i$, ω is a nonempty string of terminals of G_R, V_{β_i} and $V_{\beta_{i_1}}, ..., V_{\beta_{i_{k-1}}}$ are nonterminals of G_R, and V_{β_j} is a nonterminal of G_R or is ϵ. Notice that $\phi_1, ..., \phi_{k-1}$ are nonempty since every right-linear production has a nonempty *terminal string*.[17]

(1) Termination: To see that the algorithm halts with a grammar G_R only polynomially larger than G_P, observe that the terminal string in each new production is a suffix of some α_i or γ_i in G_P, and no new non-terminal symbols are introduced. Let α be the longest of $\alpha_1, ..., \alpha_n$, let β be the longest of $\beta_1, ..., \beta_m$, and let γ be the longest of $\gamma_1, ..., \gamma_m$. It follows that at most $n|\alpha| + m|\gamma|$ such suffixes exist. Thus there are only $m(m+1)(n|\alpha| + m|\gamma|)$ productions[18] added to G_R, although the algorithm may take exponential time to construct this grammar. (We are interested in the algorithm only to establish an equivalence between the languages and to describe the sizes of their representations; thus it need not be efficient.)

(2) $L(G_P) = L(G_R)$: The intuition behind the algorithm is helpful in understanding why $L(G_R) = L(G_P)$. In presenting the intuition, and in the proof, we give the nonterminal S of G_R the alternative name "V_ϵ". Note that this does not create a naming conflict since ϵ never appears as the left-hand side of a production of G_P. The strings α_i, $1 \leq i \leq n$, in G_P each assert, "α_i is a string in $L(G_P)$." The productions $\beta_j \rightarrow \gamma_j$, $1 \leq j \leq m$, in G_P assert, "if $\beta_j\omega$ is a string in $L(G_P)$ for some string ω, then $\gamma_j\omega$ is also in $L(G_P)$," or less formally, "anything that can follow β_j can also follow γ_j." The right-linear grammar G_R simply makes these same assertions in a different way. We want a nonterminal V_β to represent the set of all strings that can follow β in $L(G_P)$; that is, we want the set of strings that G_R can generate from V_β to be the set of all strings ω such that $\beta\omega \in L(G_P)$. As a special case, notice that we want V_ϵ or

[17] We use the phrase *terminal string* to mean the entire string of terminals on the right-hand side of a production.

[18] It can be shown that the maximum size of any such production is bounded by a polynomial in the size of G_P.

S, to represent the set of all strings that can follow ϵ in $L(G_P)$, that is, the set of all strings in $L(G_P)$. This shows $L(G_R) = L(G_P)$. To achieve this result, we want a production $V_\beta \rightarrow \gamma$ in G_R to assert, "γ can follow β in $L(G_P)$," and we want a production $V_{\beta_1} \rightarrow \gamma V_{\beta_2}$ to assert, "γ, followed by any string that can follow β_2 in $L(G_P)$, can follow β_1 in $L(G_P)$." The following lemma says this is indeed what the productions of G_R assert. The lemma is then used to prove that $L(G_R) = L(G_P)$ by showing that the productions force the nonterminals to represent the appropriate sets of strings.

Lemma 85 *At any time during the construction of G_R, (1) if $V_\beta \rightarrow \gamma$ is added to G_R then we have $\beta\gamma \in L(G_P)$, and (2) if $V_\beta \rightarrow \gamma V_{\beta'}$ is added to G_R then for any $\beta'\omega \in L(G_P)$ we have $\beta\gamma\omega \in L(G_P)$.*

Proof: All the V_ϵ-productions are added first. The result clearly holds after each of these is added: a production $V_\epsilon \rightarrow \gamma$ is added only if γ is a production of G_P, so $\gamma = \epsilon\gamma \in L(G_P)$, and a production $V_\epsilon \rightarrow \gamma V_{\beta'}$ is added only if $\beta' \rightarrow \gamma$ is a production of G_P, so if $\beta'\omega \in L(G_P)$ then applying $\beta' \rightarrow \gamma$ yields $\gamma\omega = \epsilon\gamma\omega \in L(G_P)$. Assume all productions added up to some arbitrary other point in the construction of G_R have the specified properties. If the algorithm now adds a production $V_\beta \rightarrow \gamma$, G_R must already contain productions

$$\begin{aligned} V_\epsilon &\rightarrow \phi_1 V_{\beta_{i_1}} \\ V_{\beta_{i_1}} &\rightarrow \phi_2 V_{\beta_{i_2}} \\ &\vdots \\ V_{\beta_{i_{k-1}}} &\rightarrow \phi_k \gamma \end{aligned}$$

where $\phi_1\phi_2...\phi_k = \beta$. So $\beta_{i_{k-1}}\phi_k\gamma \in L(G_P)$ and $\beta_{i_{k-2}}\phi_{k-1}\phi_k\gamma \in L(G_P)$ and ... and $\epsilon\phi_1\phi_2...\phi_k\gamma = \beta\gamma \in L(G_P)$.

Similarly, if the algorithm now adds a production $V_\beta \rightarrow \gamma V_{\beta'}$, then G_R must already contain productions

$$\begin{aligned} V_\epsilon &\rightarrow \phi_1 V_{\beta_{i_1}} \\ V_{\beta_{i_1}} &\rightarrow \phi_2 V_{\beta_{i_2}} \end{aligned}$$

$$V_{\beta_{i_{k-1}}} \rightarrow \phi_k \gamma V_{\beta'}$$

where $\phi_1\phi_2...\phi_k = \beta$. Then for any $\beta'\omega \in L(G_P)$, it is the case that $\beta_{i_{k-1}}\phi_k\gamma\omega \in L(G_P)$ and $\beta_{i_{k-2}}\phi_{k-1}\phi_k\gamma\omega \in L(G_P)$ and ... and $\epsilon\phi_1\phi_2...\phi_k\gamma\omega = \beta\gamma\omega \in L(G_P)$. □

(2.1) $L(G_P) \supseteq L(G_R)$:

Proof: $L(G_P) \supseteq L(G_R)$ may be restated as follows: if $V_\epsilon \stackrel{*}{\Rightarrow} \gamma$ is a derivation of G_R then $\epsilon\gamma = \gamma \in L(G_P)$. We prove, more generally, that for any nonterminal V_β of G_R, if $V_\beta \stackrel{*}{\Rightarrow} \gamma$ is a derivation of G_R then $\beta\gamma \in L(G_P)$, and if $V_\beta \stackrel{*}{\Rightarrow} \gamma V_{\beta'}$ is a derivation of G_R then for any $\beta'\omega \in L(G_P)$ we have $\beta\gamma\omega \in L(G_P)$. The proof is by induction on the length of the derivation. If the length is 1, then $V_\beta \rightarrow \gamma$ ($V_\beta \rightarrow \gamma V_{\beta'}$) is a production of G_R, and the result therefore holds by Lemma 85. If the length is greater than 1, let $V_\beta \rightarrow \gamma_1 V_{\beta''}$ be the first production applied in the derivation. By Lemma 85, for any $\beta''\omega \in L(G_P)$ we have $\beta\gamma_1\omega \in L(G_P)$. Then the remainder of the derivation must be $\gamma_1 V_{\beta''} \stackrel{*}{\Rightarrow} \gamma_1\gamma_1$ (or $\gamma_1 V_{\beta''} \stackrel{*}{\Rightarrow} \gamma_1\gamma_1 V_{\beta'}$) where $\gamma_1\gamma_2 = \gamma$. This derivation implies the existence of a derivation $V_{\beta''} \stackrel{*}{\Rightarrow} \gamma_2$ (or $V_{\beta''} \stackrel{*}{\Rightarrow} \gamma_2 V_{\beta'}$) of the same length. By the inductive hypothesis, $V_{\beta''} \stackrel{*}{\Rightarrow} \gamma_2$ implies $\beta''\gamma_2 \in L(G_P)$ (and $V_{\beta''} \stackrel{*}{\Rightarrow} \gamma_2 V_{\beta'}$ implies that for any $\beta'\omega' \in L(G_P)$ we have $\beta''\gamma_2\omega' \in L(G_P)$). Therefore, because if $\beta''\omega \in L(G_P)$ then $\beta\gamma_1\omega \in L(G_P)$, letting ω be γ_2 (or $\gamma_2\omega'$) we have $\beta\gamma_1\gamma_2 = \beta\gamma \in L(G_P)$ (or $\beta\gamma_1\gamma_2\omega' = \beta\gamma\omega' \in L(G_P)$). □

(2.2) $L(G_P) \subseteq L(G_R)$:

Proof: We begin by proving that for any nonterminal V_β other than V_ϵ, if G_R generates γV_β and G_P generates $\beta\omega$ then G_R generates $\gamma\omega$. The proof is by induction on the length of the derivation of γV_β in G_R. If the length of the derivation is 1, G_R must contain a production $V_\epsilon \rightarrow \gamma V_\beta$. Then $\beta \rightarrow \gamma$ must be a production of G_P. Therefore, since G_P generates $\beta\omega$ and has a production $\beta \rightarrow \gamma$, it generates $\gamma\omega$ as well. If the length of the derivation of γV_β in G_R is greater than 1 then, because all productions in G_R are right-linear, the last sentential form generated before γV_β in a given derivation is some $\gamma_1 V_{\beta'}$, where $\gamma = \gamma_1\gamma_2$, γ_2 is nonempty, and

$V_{\beta'} \rightarrow \gamma_2 V_\beta$ is a production of G_R. By Lemma 85 we know from $V_{\beta'} \rightarrow \gamma_2 V_\beta$ and $\beta\omega \in L(G_P)$ that $\beta'\gamma_2\omega \in L(G_P)$. By the inductive hypothesis, since G_R generates $\gamma_1 V_{\beta'}$ (with a shorter derivation than for γV_β) and $\beta'\gamma_2\omega \in L(G_P)$, we know G_R generates $\gamma_1\gamma_2\omega = \gamma\omega$.

To prove the complete result, let ω be any string that G_P generates. If the derivation of ω has length 1, that is, if ω is itself a production of G_P, then G_R has a production $V_\epsilon \rightarrow \omega$, so clearly G_R generates ω. Otherwise, the last production applied in the derivation of ω is some $\beta \rightarrow \gamma$ that maps $\beta\gamma'$ to $\gamma\gamma'$ where $\omega = \gamma\gamma'$. Because G_P has the production $\beta \rightarrow \gamma$, G_R has the production $V_\epsilon \rightarrow \gamma V_\beta$. Since G_R generates γV_β and G_P generates $\beta\gamma'$, by our previous result we know that G_R generates $\gamma\gamma' = \omega$. □

6.7 The Prefix Languages Contain the Regular Languages

input: dfa $M = (Q, q_0, \Sigma, F, \delta)$
output: prefix grammar G

1. For each $f \in F$
   (a) For each string γ accepted by a cycle–free transition sequence from q_0 to f.
   (b) Add γ to the base set S of G
2. For each $q \in Q$
   (a) For each pair of strings (α, β) each of which induce a cycle–free transition sequence from q_0 to q
      i. Add the production $\alpha \rightarrow \beta$ to G
      ii. Add the production $\beta \rightarrow \alpha$ to G
   (b) For each pair of strings (α, β) where α induces a cycle–free transition sequence from q_0 to q and β induces a cycle from q to q which does not properly contain any cycles
      i. Add the production $\alpha \rightarrow \alpha\beta$ to G

Claim 86 *The algorithm produces a prefix grammar that defines exactly the same language as the given DFA.*

Proof: First observe that at each step the algorithm looks for cycle-free transition sequences. Such sequences are bounded in length by $|Q|$, and hence there are at most $|Q|!$ such sequences. Thus the algorithm terminates.

Next observe that for DFAs, a given string and an initial state uniquely determine the state after applying the string to the initial state. Now, every production in G replaces a prefix of a sentential form with another prefix, and the replacement prefixes are chosen from among those strings which leave the DFA in precisely the same state as the replaced prefix. Thus every production of the grammar is guaranteed to preserve acceptance. Finally, the only sentential forms explicitly given for $L(G)$ are strings accepted by the DFA. Thus $L(G) \subseteq L(M)$.

What now remains is to show that $L(M) \subseteq L(G)$. This portion of the proof is accomplished by strong induction on the number of state revisitations[19] by a string accepted by the DFA. Suppose w is accepted by M and no state revisitations occur. In this case, this string was explicitly placed in the grammar G in step 1 of the algorithm; thus $w \in L(G)$. Now suppose inductively that all strings accepted by M causing at most k state revisitations can be generated by G, and consider any string w accepted by M which causes $k+1$ state revisitations. Necessarily, w must cause M to revisit some state within its first $|Q|$ symbols. Furthermore, without loss of generality, we may assume that this revisitation is a simple-cycle. Consider the first such revisit. Name the substring that causes this revisitation w_m so that w may be rewritten as $w_p w_m w_s$ (where w_p is the prefix of w, w_m is the middle of w, and w_s is the suffix of w). We can remove w_m from w and the resulting string $w_p w_s$ is accepted by M and has at most k state revisitations, so inductively, $w_p w_s$ can be generated by G. Now, by our choice of w_m, w_p induces a cycle-free transition from q_0 to some state q and w_m induces a cycle from q to q which does not itself properly contain any cycles. But this implies that step 2 of the algorithm added the production $w_p \rightarrow w_p w_m$ to G. Thus we can apply this production to the string $w_p w_s$ to obtain $w_p w_m w_s = w$. Therefore $w \in L(G)$, which completes the induction and the proof. □

This completes the proof that the regular languages are contained in the prefix languages.

[19] A state revisitation occurs whenever any state that has already been visited is visited again. Thus a string which produces the state sequence q_0, q_4, q_3, q_0, q_3 causes 2 state revisitations – 1 revisitation of the state q_0 and one revisitation of the state q_3.

6.8 Applications to Learnability

It follows from Theorem 84 that $\mathcal{H}_{*,1}$ can be learned in *exponential* time *in terms of* DFAs with equivalence queries *and membership queries*[20] using Angluin's [3] L^* algorithm.[21] We should emphasize that this result does not subsume the learnability result given earlier for $\mathcal{H}_{2,1}$ (it clearly does not subsume the result for $\mathcal{H}_{2,k}$) for at least two reasons. First, the earlier result is for learning in polynomial time. Second, the earlier result does not require membership queries.

An interesting open question is whether there exists a polynomial time transformation from strings to strings such that for every prefix grammar there exists a DFA, rather than right-linear grammar, only polynomially larger than the prefix grammar that accepts the transformation of language generated by the grammar.[22] If so, $\mathcal{H}_{*,1}$ can be learned in terms if DFAs, and therefore predicted, with membership and equivalence queries. If, on the other hand, it can be shown that there exists a polynomial size prefix grammar for any NFA, then predicting $\mathcal{H}_{*,1}$ is as hard as predicting NFAs, which cannot be done (under certain cryptographic assumptions) [10, 66].

6.9 Related Work

The concept classes studied in this chapter are incomparable to—that is, neither subsume nor are subsumed by—other classes of definite clause theories whose learnability, in the PAC or exact models, has been investigated [41, 82].[23] Page and Frisch [82] investigated classes of definite clauses that may have predicates and functions of arbitrary arity but explicitly do *not* have recursion. In that work, a *background* theory[24] was also allowed; allowing such a theory in the present work is an interesting topic for future research. Cohen [35] investigates the learnability of function-free, two clause, closed, linearly recursive, ij-determinant logic programs, given a background theory. His positive result appeared at the same time as this chapter [43], and

[20] A membership query asks whether the target concept classifies a particular example as positive or negative.

[21] We say in this case that $\mathcal{H}_{2,*}$ can be *predicted* using membership and equivalence queries.

[22] It can be shown that if we demand that the transformation be the identity transformation, i.e., that the DFA accept exactly the set of strings generated by the grammar, then the answer to this question is "no".

[23] Theoretical work related to the learnability of definite clauses in other models includes Shapiro's [95] work on learning in the limit, Ling's [70] investigation of learning from *good examples*, and the work in Chapters 2 and 3 of this thesis on propositional Horn clause theories.

[24] A background theory essentially represents what the learner already knows about the world.

both are distinctive in that they seem to be the first positive learnability results for recursive theories for which the learner is not allowed to ask about the classification of specific examples. Cohen [34] gives a negative result when the linear recursiveness restriction is removed and the target consists of a single clause.

Džeroski, Muggleton, and Russell [41] investigated the learnability of classes of function-free determinate k-clause definite clause theories under simple distributions, also in the presence of a background theory.[25] This class includes recursive concepts; to learn recursive concepts, the algorithm requires two additional kinds of queries (*existential queries* and *membership queries*). Rewriting definite clause theories that contain functions to function-free clauses allows their algorithm to learn in the presence of functions. Nevertheless, the restriction that clauses be *determinate* effectively limits the depth of function nesting; their algorithm takes time exponential in this depth. So, for example, while the algorithm can easily learn the concept *even integer*, or *multiple of 2*, from $\mathcal{H}_{2,1}$ —$[even(0)] \wedge [even(x) \rightarrow even(s(s(x)))]$—the time it requires grows exponentially in moving to a concept such as *multiple of 10* or *multiple of 1000*, also in $\mathcal{H}_{2,1}$. It is easy to show that the classes $\mathcal{H}_{2,1}$, $\mathcal{H}_{*,1}$, $\mathcal{H}_{2,k}$, and $\mathcal{H}_{2,*}$, rewritten to be function-free, are not $\{i,j\}$-determinate, for any i and j.

Finally, Džeroski, Muggleton, and Russell [41] obtain their results by a transformation to a concept class in propositional logic, and they note the importance of finding learnable classes of first-order concepts that have no such transformation. While we have been unable to find any such transformation for the classes presented here, showing that a class has *no* such transformation appears to be a difficult task, and also one we have been unable to accomplish. In addition, Section 6 indicates that it may be important as well to determine whether a learnable class of first-order concepts has such a transformation to a class of formal languages or automata, particularly when recursion is present. That section also leads one to speculate about whether various classes of first-order concepts may even be interesting hybrids between concepts of propositional logic and formal languages or automata. The relationship of concept classes described in this chapter to other classes is also an interesting area for further research.

[25] The restriction to simple distributions is a small one that quite possibly can be removed.

Chapter 7

Description Logics

Description logics, also called *terminological logics*, are commonly used in knowledge-based systems to describe objects and their relationships. This chapter investigates the learnability of a typical description logic, CLASSIC, and shows that CLASSIC sentences are learnable in polynomial time in the exact learning model using membership queries (which are in essence, "subsumption queries"). It is shown that membership queries alone are insufficient for polynomial time learning of CLASSIC sentences. Combined with earlier negative results of Cohen and Hirsh [33] showing, given standard complexity-theoretic assumptions, that random examples alone are insufficient in the PAC setting, this shows that both sources of information are necessary for efficient learning in that neither type alone is sufficient. In addition, it is shown that a modification of the algorithm deals robustly with persistent malicious two-sided classification noise in the membership queries with the probability of a misclassification polynomially bounded below 1/2. This fact that this first order language can be learned in full is surprising; even more surprising is the robust error tolerance.

7.1 Introduction

A central problem facing automated Information Systems is knowledge acquisition (eg. [76]). Whether dealing with expert systems, database constraints, or deductive databases, applying knowledge demands that knowledge be at hand. Unfortunately, extracting knowledge from a domain expert results in either an extremely narrow base of knowledge, or an enormous

amount of potentially buggy knowledge, or both. We address the problem of efficient knowledge acquisition from the vantage point of computational learning theory.

Traditionally, computational learning theory has focused on propositional domains. We investigate learning in the first-order domain of *description logics* or *terminological logics*. Specifically we consider the learnability of the description logic known as CLASSIC [39]. To the extent that CLASSIC is a typical description logic, our results generalize to a variety of other such logics.

Description logics are more expressive than the propositional calculus. A description logic statement is essentially a first-order predicate calculus formula in which all but one variable is quantified. Therefore, the meaning of a statement in a description logic, instead of being either true or false for a given interpretation, is the subset of the universe satisfying the statement. For example, suppose that the universe is a set of dogs, $\texttt{brown}(x)$ asserts that x is brown, and $\texttt{smaller}(x,y)$ asserts that y is smaller than x. If it happens to be the case that **Rex** is the only shaggy dog and **Fido** is the only brown dog, then $\forall y \ \texttt{brown}(x) \wedge \texttt{smaller}(x,y)$ is a well-formed description logic statement denoting the set {**Fido**} provided **Fido** is the largest dog in the universe; otherwise the empty set is denoted. Likewise $\texttt{brown}(x) \wedge \texttt{shaggy}(x)$ denotes the empty set. Neither statement is a closed formula in the predicate calculus and neither statement has an associated truth value. Thus, description logics have a different flavor than the predicate calculus. Description logics comprise "natural" classes of formulas in that description logics are used in the field of knowledge representation [17, 28, 25, 26, 39, 68, 77, 83].

7.1.1 CLASSIC

Essentially, CLASSIC permits constructing certain quantified descriptions which distinguish a particular subset of a domain I of *individuals*. CLASSIC descriptions contain *primitive symbols* which get mapped to arbitrary subsets of I, *disjoint primitive symbols* which get mapped to mutually disjoint subsets of I, *roles* which get mapped to binary relations on I, and *attributes* which are roles that happen to be functions. Further, CLASSIC sentences contain constructors which manipulate these primitives, roles, and attributes, in order to permit the denotation of complicated subsets of I. The following synopsis and semantics of CLASSIC is excerpted from [26, 36, 33].

(SAME-AS $(r_{1,1} \ldots r_{1,k_1})$ $(r_{2,1} \ldots r_{2,k_2})$**)** denotes the set

$$\{x : r_{1,k_1}(\cdots(r_{1,2}(r_{1,1}(x)))) = r_{2,k_2}(\cdots(r_{2,2}(r_{2,1}(x))))\}$$

of individuals for which composing the first chain of attributes is the same as composing the second chain of attributes.

(ALL r D**)** denotes the set $\{x : \forall y\,[r(x,y) \rightarrow D(y)]\}$ of individuals for which *all* of the r-related individuals satisfy description D.

(AND $D_1 \ldots D_n$**)** denotes the set $\{x : D_1(x) \wedge \cdots \wedge D_n(x)\}$ of individuals that satisfy all of the descriptions $D_1, \ldots, D_n$.

(AT-LEAST n r**)** denotes the set $\{x : |\{y : r(x,y)\}| \geq n\}$ of individuals having at least n r-related individuals.

(AT-MOST n r**)** denotes the set $\{x : |\{y : r(x,y)\}| \leq n\}$ of individuals having at most n r-related individuals.

(PRIM p_i**)** denotes the subset of individuals denoted by the primitive symbol p_i (provided by the interpretation).[1]

(FILLS r $p_1 \ldots p_n$**)** denotes the set $\{x : \exists y_i \in p_i$ such that $r(x,y_1) \wedge \cdots \wedge r(x,y_n)\}$, where the p_i are disjoint primitive symbols.

(ONE-OF $p_1 \ldots p_n$**)** denotes the set $\cup_{i=1}^{n} p_i$, where the p_i are disjoint primitive symbols.

Descriptions are built from the individuals, primitives, and other descriptions. For example if our set of individuals is the set of all dogs and breeds[2] and we have at our disposal the primitive concept **brown** for the set of brown dogs, the role **smaller** for comparing dog sizes, the attribute **breed** for denoting breeds, the attribute **father** for associating a dog with its

[1] In our illustrations we omit this formalism and use descriptions such as **brown** to denote the primitive set of things which are brown.

[2] Keep in mind that these two different "types" of individuals are indistinguishable within a description logic statement; it is the venue of the externally supplied meanings of the roles and primitives to preserve any intuitive distinctions we may have concerning these different "types" of individuals.

father, the attribute **mother** for associating a dog with its mother, and the role **school** denoting the obedience school classmates, then if we wished to denote the set

$$\{x : \forall y\ \mathbf{school}(x,y) \rightarrow [\mathbf{brown}(y) \wedge |\{z : \mathbf{smaller}(y,z)\}| \geq 20 \wedge \mathbf{breed}(\mathbf{mother}(y)) = \mathbf{breed}(\mathbf{father}(\mathbf{father}(y)))\ ]\ \}$$

of dogs *all* of whose obedience school classmates were brown, larger than at least twenty other dogs, and had mother and paternal grandfather of the same breed, we would write:

(ALL **school** (AND **brown**
(AT-LEAST 20 **smaller**)
(SAME-AS (**mother breed**) (**father father breed**))))

7.1.2 The Learning Problem

The meaning of a description logic statement depends on a particular interpretation. It is a set selector: Given a choice of a universal set of individuals I, an assignment of the primitive symbols (such as **brown**) to subsets of I, an assignment of the roles to binary relations on I, and an assignment of attributes to functions from I to I, the statement denotes the set of elements x in I that cause the corresponding first order expression to evaluate to true, given the semantics on the previous page.

One way to define a positive example of a CLASSIC sentence is as an individual that is in the denotation of the sentence. One drawback of this definition is that the classification of an example depends on a particular interpretation, and not just on the target concept. Further, the classification may depend on the relationships among a potentially infinite number of individuals from the universe. An alternative definition, taken by Cohen and Hirsh [33], and supported in the description logic community [25, 24, 40], is to define a positive example to be the description of an individual (using the same description logic) that is a positive example for every possible interpretation. Because such a description may not select a unique individual for all interpretations, each such (positive) example is actually a concept itself, whose denotation is a subset of the denotation of the target concept. Thus, C is a positive example if C_* *subsumes* C. This viewpoint is also supported by previous work in inductive logic programming [82, 41, 43]

and learning from "entailment" [46, 2], where positive examples of an unknown (typically first-order) formula are clauses or other formulas that are entailed by the unknown formula.

We employ the standard protocol of learning from equivalence and membership queries. Our algorithm may conjecture any CLASSIC description H, and is told whether or not H is equivalent to the target description C_* (i.e., has the same denotation for all possible interpretations). If H is not equivalent to C_*, then the algorithm receives a counterexample, which is a positive example of one of H and C_*, but not both. The algorithm also may present a description C, and is then told whether or not C is a positive example of C_* (i.e., whether or not C_* subsumes C). Of course, a standard transformation turns this algorithm into a PAC with membership query learning algorithm.

The algorithm takes advantage of the fact that such queries C may be arbitrary concepts, so perhaps it is more appropriate to call these "subsumption" queries, or even "subset" queries. The distinction between these notions is lost given the common use of the "single representation trick" in AI - where it is often convenient and desirable to represent concepts and examples using the same language. In fact, Cohen and Hirsh [33] note that in many implemented description logics, "it is possible to attach an arbitrary description to an instance [example], hence the distinction between instances [examples] and concepts is blurred."

Because the subsumption relation is computable in polynomial time [36, 33], both membership and equivalence queries are efficiently computable by a teacher. The main results of the chapter are now summarized:

Theorem CLASSIC is learnable in time polynomial in the size of the target description and the length of the longest counterexample, using membership (subsumption) and equivalence queries.

Theorem Any algorithm using membership (subsumption) queries alone requires a number of membership queries that is exponential in the size of the target concept.[3]

[3]Thus the positive result does not come solely from the membership queries. Cohen and Hirsh [33] showed that CLASSIC is not learnable in polynomial time (without membership queries) in the PAC model (assuming RP $\neq$ NP), hence CLASSIC cannot be learned from equivalence queries alone given the same assumption. Thus, neither membership nor equivalence queries are dispensable - they form a minimal set of learning queries for CLASSIC.

Theorem CLASSIC remains learnable in the exact learning model with equivalence and membership queries, even when each answer to a membership query may be answered incorrectly by a malicious adversary with probability $1/2 - 1/r$, where r is any polynomial function of the size of the target concept.[4]

In particular, when a membership query is asked, an adversary flips coins, and with probability $1/2 - 1/r$, may choose to classify incorrectly. (Any future queries on the same example are results in the same classification). We present a modification of our algorithm that will work when r is any polynomial function of the size of the target concept. To our knowledge, this is the first algorithm for any concept class capable of coping robustly with such errors.

7.1.3 Comparison to Previous Results

Automating propositional explanation discovery has been well studied [7]. In comparison, efficient first-order learnability has been less well studied. Even so, some results are known. Cohen [35] gives a PAC learning algorithm for function-free, two clause, closed, linearly recursive, ij-determinant logic programs; he goes on to show [34] that when the condition linear recursiveness is relaxed, the learning problem becomes cryptographically hard. Page and Frisch [82] show that constrained atoms (a typed logic) are efficiently learnable, Frazier and Page [43] provide a learning algorithm for a syntactically restricted subclass of first-order Horn formulas, and Džeroski et al. [41] provide a learning algorithm for a different restriction of first-order Horn formulas.

Haussler [57] investigated existentially quantified conjunctive concepts and describes a graph representation for those concepts. He showed that learning some very simple scene descriptions is difficult. Specifically, he showed that even restricted to unary atoms such concepts are not learnable from random examples unless $RP = NP$, but did give a learning algorithm for settings where the algorithm may use a richer vocabulary than that from which the target was chosen. Indeed, positive first-order learning results appear to be quite rare for "natural" classes of first-order formulas. It would seem that the difficulty of the learning task he faced revolved around the ambiguity admitted by the graphical representation required to capture existential

[4] The errors are *persistent*, so that the algorithm may not benefit from repeatedly asking the same question.

quantification in the concept class he investigated; our concept class does not permit existential quantification. It will be seen that the graphs we use suffer no such ambiguity, thus we are able to avoid the difficulty he faced.

The work most closely related is that of Cohen and Hirsh [33] who employ a graphical representation for CLASSIC concepts developed by Borgida and Patel-Schneider [26]. To explain their results, and present ours, we briefly explain the notion of a *labeled equivalence graph* (called a *concept graph* in [26, 36, 33]).

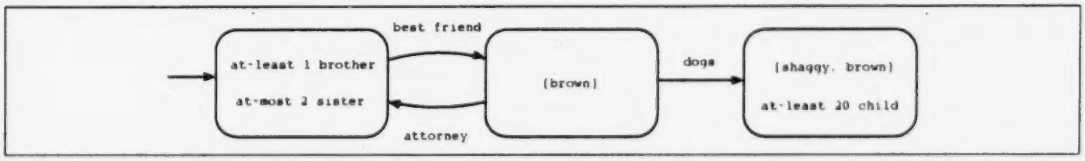

Figure 7.1: A labeled equivalence graph.

Consider the graph in figure 7.1. This is a graphical depiction of the CLASSIC description of the set of individuals who have at least one brother and at most two sisters, whose best friend has brown hair, who are their best friend's attorney, and whose best friend only has brown, shaggy dogs that have at least twenty puppies. The cycle in the graph also asserts infinitely many other SAME-AS conditions – for example, conditions about the best friend's attorney's best friend.

Formally defined in Section 3, a labeled equivalence graph is a rooted, directed, vertex- and edge-labeled graph. Further, no vertex has two identically labeled outgoing edges. The edge labels represent binary relations over the universe of individuals, and an edge demands that all individuals in the image of the relation satisfy the constraints asserted by the vertex to which the edge leads. Also, any pair of disjoint directed paths between a pair of vertices involve only binary relations which are in fact functions. This pair asserts that the individual selected along one path must be the same as the individual selected along the other path. We associate a subset of the set of individuals I with each vertex in the graph — the subset of individuals satisfying every constraint – whether vertex label, edge label, or assertion of equivalence at some reachable vertex – asserted by the graph. The set of individuals denoted by the graph is exactly the set of individuals associated with the root. Note that the presence of directed cycles in the graph, and in particular, those involving the root, implies that the root concept is

being defined in terms of other concepts, ..., which are in turn being defined in terms of the root. Thus, cycles allow co-referential definitions.

Polynomial time algorithms exist for transforming labeled equivalence graphs into CLASSIC sentences, and vice-versa [26, 36, 33], although not all labeled equivalence graphs correspond to valid CLASSIC sentences. Thus, the question of learnability of CLASSIC sentences more or less reduces to that of learning such equivalence graphs. What would a positive example of an equivalence graph look like? It is another graph which satisfies all of the constraints (and perhaps more) represented by the first. The subsumption algorithm for CLASSIC essentially verifies that the vertex label reached by a path in the first graph is less restrictive than the label of the corresponding vertex (which must exist) in the second graph, and that if in the first graph the two paths labeled by strings w_1 and w_2 lead to the same vertex, then this occurs in the second graph as well. For example, if we add an edge or vertex label to the graph in Figure 7.1 we obtain a positive example. Conversely, deleting an edge or vertex label from Figure 7.1 produces a negative example. The hard part of the learning problem is to determine the structure of the graph and the edge labels, not to determine the vertex labels. Thus, most of the constructors from the CLASSIC language are not problematic; the main challenge is presented by the SAME-AS conditions (each represented by a disjoint pair of paths between two vertices), and the role and attributes (which are edge labels). Henceforth we assume that all the vertex labels are irrelevant; in Section 7.4 we show how the algorithm is modified when this is not the case.

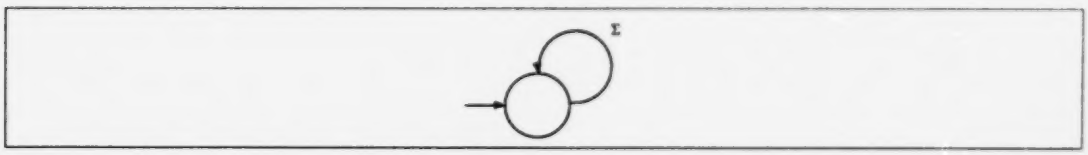

Figure 7.2: The universally positive example for equivalence graphs. Σ indicates that every possible edge label $\sigma \in \Sigma$ appears on a directed edge from the root vertex back to itself.

A natural first attempt to learn such graphs would be to simply intersect the graphs which represent positive examples, thereby extracting the set of common vertices and edges. However, since the universal positive example (figure 7.2) contains *every* path (and thus is subsumed by *any* target), simply intersecting the graphs will not be enough. What does succeed is a variant of the "cross-product of DFAs" construction for regular language intersection. By repeatedly

taking the cross-product of positive examples, a (one-sided) learning algorithm is obtained. A problem with this approach is that the cross-product of two equivalence graphs can be as large as the product of their sizes; the repeated cross-product necessary to implement this approach may yield exponentially sized hypotheses.

Cohen and Hirsh [33] circumvent this problem by restricting the number of distinct paths through the graphical representation of a CLASSIC concept. Given a constant k they consider graphs G having at most $|G|^k$ distinct paths (hence their graphs are acyclic). Denote this class k-CLASSIC. They show that the intersection approach above yields a $O(m^{k+1})$ mistake-bounded one-sided learning algorithm for k-CLASSIC, assuming all counterexamples have size at most m. Negatively, they show that in the PAC learning model, assuming that RP $\neq$ NP, CLASSIC is not learnable from random examples alone, even if either of the following constraints hold: (i) the primitive class alphabet is singleton, the role alphabet is doubleton, and the equivalence graph of every example is acyclic, or (ii) the primitive class alphabet is singleton, and the equivalence graph of every example contains only two vertices.

7.2 The Algorithm

Simply stated, our algorithm employs the one-sided approach of graph cross-product discussed above, but uses membership queries to bound the intermediate hypothesis size. Figures 7.3 and 7.4 give the learning algorithm. The cross-product $G \times H$ of labeled equivalence graphs G and H is described in the next section, as is the argument that the algorithm efficiently learns. At first glance, equivalence graphs seem DFA-like, but their semantics are quite different, so well-known DFA learning algorithms [3, 89] do not apply.

```
Learn
1   Let H be the universally positive example
2   H := Prune(H)
3   While EQUIVALENT(H) provides counterexample G
4     G := Prune(G)
5     H := Prune(G × H)
6   Return H
```

Figure 7.3: Equivalence graphs learning algorithm.

Prune(G)

/* G is a positive example. */

1 For each edge e in G

2 If **MEMBER**($G \backslash e$)[a] is "yes", then remove e from G

3 Return G.

[a]Along with e remove any unreachable component.

Figure 7.4: Algorithm using membership queries to remove excess graph elements from a positive example.

7.3 Equivalence Graphs

As discussed above, the learning problem for CLASSIC sentences is closely related to that of learning labeled equivalence graphs. We first consider equivalence graphs without vertex labels, and then indicate how the algorithm is modified to the more general case in Section 7.4. Later, in Section 7.5, we modify the algorithm again in order to learn CLASSIC.

Definition 87 *Let Σ be a finite alphabet. A rooted, directed, edge-labeled graph G is an* equivalence graph over Σ *if each vertex v in G is reachable from the root and for every symbol σ in Σ, v has at most one outgoing edge labeled σ. The* size $|G|$ *of an equivalence graph is the sum of the number of edges and vertices in G.*

A string w of Σ^* is *G-supported* if w is the concatenation of symbols on the edges of a rooted, directed path in G. G defines an equivalence relation $\equiv_G$ on strings of Σ^* as follows: $w_1 \equiv_G w_2$ iff both w_1 and w_2 are G-supported, and their paths terminate at the same vertex. Thus, a G-unsupported string is not G-equivalent to any other string. The set of all strings G-equivalent to a string w is denoted $[w]_G$, and, by an abuse of notation, the set of all strings that terminate at a vertex v of G is denoted $[v]_G$. It is easily verified that for any equivalence graph G, $\equiv_G$ is a right-invariant equivalence relation on strings, and that if w is G-supported then so is every prefix of w.

We define a partial order on equivalence graphs based on which strings are supported and which strings are equivalent:

Definition 88 *G_1* subsumes *G_2 if every G_1-supported string is G_2-supported and every pair of G_1-equivalent strings are G_2-equivalent.*

Examples will be labeled according to their relationship to the target under this partial ordering. Positive examples of an equivalence graph G will be exactly those graphs G' which are subsumed by G.

Definition 89 *([33]) Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two equivalence graphs. The* cross-product *of G_1 and G_2, denoted $G_1 \times G_2$, is defined as follows. Let $p_1, p_2, \ldots, p_{|V_1|}$ denote the vertices of G_1 with p_1 denoting the root, and let $q_1, q_2, \ldots, q_{|V_2|}$ denote the vertices of G_2 with q_1 denoting the root. If G_1 or G_2 is empty, then $G_1 \times G_2$ is empty. Otherwise, the vertex set of $G_1 \times G_2$ (a subset of $V_1 \times V_2$) and the edge set of $G_1 \times G_2$ are defined recursively:*

- *The graph $G_1 \times G_2$ has a root denoted (p_1, q_1).*
- *The graph $G_1 \times G_2$ has a vertex denoted (p_{i_2}, q_{j_2}) and edge $(p_{i_1}, q_{j_1}) \xrightarrow{\sigma} (p_{i_2}, q_{j_2})$ iff G_1 has edge $p_{i_1} \xrightarrow{\sigma} p_{i_2}$ and G_2 has edge $q_{j_1} \xrightarrow{\sigma} q_{j_2}$ and $G_1 \times G_2$ has the vertex denoted (p_{i_1}, q_{j_1}).*

Note that $G_1 \times G_2$ is an equivalence graph whenever both G_1 and G_2 are equivalence graphs. The following properties of $G_1 \times G_2$ are either easily verified, or follow from [36].

Property 90

1. *A string w is $(G_1 \times G_2)$-supported iff w is both G_1-supported and G_2-supported.*
2. *For any strings s and t, $s \equiv_{G_1 \times G_2} t$ iff both $s \equiv_{G_1} t$ and $s \equiv_{G_2} t$.*
3. *$G_1 \times G_2$ is the most specific generalization (least upper bound) of G_1 and G_2 with respect to the subsumption ordering. That is, if G_1, G_2, and G are equivalence graphs, then if G subsumes both G_1 and G_2, then G subsumes $G_1 \times G_2$.*

Definition 91 *An equivalence graph G is said to be* pruned with respect to *an equivalence graph G_* if G_* subsumes G, but does not subsume any proper subgraph of G.*

The following is a useful property of pruned graphs.

Property 92 *Let G and G_* be two equivalence graphs such that G is pruned with respect to G_*. Then for every vertex v in G and every outgoing edge label σ from v, $[v]_G$ contains some (G_*-supported) string s such that $s\sigma$ is G_*-supported.*

Proof Sketch: Suppose to the contrary that G contains a vertex v with outgoing edge label σ such that $[v]_G$ contains no string s such that $s\sigma$ is G_*-supported. But then deleting v's outgoing edge labeled σ produces a proper subgraph of G that supports every G_*-supported string and leaves equivalent every pair of G_*-equivalent strings. This contradicts the hypothesis that G is pruned. □

We need the following technical lemma.

Lemma 93 *Let G_1, G_2, and G_* be equivalence graphs such that both G_1 and G_2 are pruned with respect to G_*. If there exists a vertex v of $G_1 \times G_2$ such that $[v]_{G_1 \times G_2}$ contains only G_*-unsupported strings, then there are two G_*-supported strings that are G_1-equivalent but not $(G_1 \times G_2)$-equivalent.*

Proof: First observe that if G_* supports no strings at all – not even the empty string – then G_*, G_1, and G_2 must each be the empty graph. In this case the lemma holds trivially. The rest of the proof assumes that at least the empty string is supported by G_* and therefore also by G_1 and G_2 by Property 90 item 1. The proof assumes the existence of a vertex v of $G_1 \times G_2$ such that $[v]_{G_1 \times G_2}$ contains only G_*-unsupported strings and then constructs two G_*-supported strings s and s' that are G_1-equivalent but $(G_1 \times G_2)$-inequivalent.

Let v be a vertex of $G_1 \times G_2$ such that $[v]_{G_1 \times G_2}$ contains only G_*-unsupported strings, and let w be any string in $[v]_{G_1 \times G_2}$. Now, since the $G_1 \times G_2$ equivalence class containing the empty string contains a G_*-supported string (the empty string itself) and since $[w]_{G_1 \times G_2}$ contains no G_*-supported stings, there exists a prefix w_p of w and an edge label σ such that

- $w_p\sigma$ is a prefix of w,
- $[w_p]_{G_1 \times G_2}$ contains a G_*-supported string, and
- $[w_p\sigma]_{G_1 \times G_2}$ contains no G_*-supported string.

Let s be any G_*-supported string in $[w_p]_{G_1 \times G_2}$. Now observe that since w is $(G_1 \times G_2)$-supported so are both $w_p\sigma$ and w_p. Also observe that since $w_p\sigma$ is $(G_1 \times G_2)$-supported $w_p\sigma$ must be G_1-supported by Property 90 item 1. But then, by Property 92 since G_1 is pruned with respect to G_*, $[w_p]_{G_1}$ must contain a G_*-supported string s' such that $s'\sigma$ is also G_*-supported. Thus we have two G_*-supported strings s and s' such that $s \equiv_{G_1 \times G_2} w_p$ (which by Property 90 item 2 implies that $s \equiv_{G_1} w_p$) and $s' \equiv_{G_1} w_p$, and so $s \equiv_{G_1} s'$. Now since $s'\sigma$ is G_*-supported,

if $s \equiv_{G_1 \times G_2} s'$ then $s' \equiv_{G_1 \times G_2} w_p$ so that $[\![w_p\sigma]\!]_{G_1 \times G_2}$ contains the $G_\bullet$-supported string $s'\sigma$, contradicting the choice of w_p and σ. Thus $s \not\equiv_{G_1 \times G_2} s'$. Therefore we have constructed two $G_\bullet$-supported strings that are G_1-equivalent but are $(G_1 \times G_2)$-inequivalent. □

The proof that the learning algorithm is correct and efficient (Theorem 96) will follow easily from a technical lemma (Lemma 95), which asserts that progress is made with each new hypothesis of the algorithm. The proof of the lemma follows the proof of Theorem 96. We first need the following definition:

Definition 94 *Let G be an equivalence graph that $G_\bullet$ subsumes. Then $\equiv_G^{G_\bullet}$ is the equivalence relation $\equiv_G$ restricted to $G_\bullet$-supported strings.*

Lemma 95 *Let G_1 and G_2 both be pruned with respect to $G_\bullet$, and let $G = \textbf{Prune}(G_1 \times G_2)$. Further, suppose that G_1 does not subsume G_2. Then $\equiv_G^{G_\bullet}$ is a proper refinement of $\equiv_{G_1}^{G_\bullet}$.*

Theorem 96 *Let $G_\bullet$ be the target concept. The algorithm* **Learn** *finds an equivalence graph equivalent to $G_\bullet$, and at no point during execution does the running time exceed a polynomial in $|\Sigma|$, $|G_\bullet|$, and the size of the largest counterexample seen so far.*

Proof Sketch: The initial hypothesis has one equivalence class. A simple inductive proof shows that every hypothesis is a positive example. By Lemma 95, each counterexample causes the number of equivalence classes over supported strings to increase by at least one. Since the number of equivalence classes in the hypothesis is bounded above by the number of equivalence classes in the target[5], the number of equivalence queries is bounded above by that number.

Finally, observe that at each step, if $\tilde{G}$ is the counterexample with the greatest number of vertices seen so far, the algorithm has made at most $|G_\bullet|^2 \cdot |\tilde{G}| \cdot |\Sigma|$ membership queries, and has run for at most a number of steps that is polynomial in $|G_\bullet|$, $|\tilde{G}|$, and $|\Sigma|$. This follows from the fact that at each step, if $\tilde{G}$ is the counterexample having the greatest number of vertices seen so far, the number of membership queries used by **Prune** on $H \times \tilde{G}$ is at most $O(|\Sigma| \cdot n_H \cdot n_{\tilde{G}})$, where n_H and $n_{\tilde{G}}$ are the number of vertices in H and $\tilde{G}$, respectively. Since n_H is bounded

[5] If some vertex of the hypothesis contained no $G_\bullet$-supported string, that vertex would have been pruned. On the other hand, if the number of vertices in the hypothesis containing $G_\bullet$-supported strings is greater than the number of vertices in $G_\bullet$, then the hypothesis is a negative example because some pair of $G_\bullet$-equivalent strings are inequivalent in the hypothesis.

above by $n_{G_\bullet}$ and since $|H| < |G_\bullet|$,[6] the number of membership queries used by a single call to **Prune** is $O(|\Sigma| \cdot |G_\bullet| \cdot |\tilde{G}|)$, where $\tilde{G}$ is the largest counterexample yet witnessed by the algorithm.

□

Proof of Lemma 95: It is sufficient to show that $\equiv^{G_\bullet}_{G_1 \times G_2}$ is a proper refinement of $\equiv^{G_\bullet}_{G_1}$, since G is obtained by pruning from $G_1 \times G_2$, and no edges or vertices are added — only deleted, hence $\equiv_G$ is a refinement of $\equiv_{G_1 \times G_2}$.

Now, $\equiv_{G_1 \times G_2}$ is a refinement of $\equiv_{G_1}$. Further, since both are subsumed by $G_\bullet$, they both support every $G_\bullet$-supported string. Hence, $\equiv^{G_\bullet}_{G_1 \times G_2}$ is a refinement of $\equiv^{G_\bullet}_{G_1}$.

If the number of equivalence classes of $\equiv^{G_\bullet}_{G_1 \times G_2}$ exceeds the number of equivalence classes of $\equiv^{G_\bullet}_{G_1}$, then the lemma is proved. Otherwise, since $\equiv^{G_\bullet}_{G_1 \times G_2}$ is a refinement of $\equiv^{G_\bullet}_{G_1}$, the number of equivalence classes must be the same, and the classes must be identical. We show that this leads to a contradiction, thus proving the lemma.

By Property 90 item 2, for any strings x and y, $x \equiv_{G_1 \times G_2} y$ iff $x \equiv_{G_1} y$ and $x \equiv_{G_2} y$, and since all three support all $G_\bullet$-supported strings, we have that for any $G_\bullet$-supported strings x and y, $x \equiv^{G_\bullet}_{G_1 \times G_2} y$ iff $x \equiv^{G_\bullet}_{G_1} y$ and $x \equiv^{G_\bullet}_{G_2} y$. This, together with our assumption that the relations $\equiv^{G_\bullet}_{G_1 \times G_2}$ and $\equiv^{G_\bullet}_{G_1}$ are identical, implies that the relations $\equiv^{G_\bullet}_{G_1}$ and $\equiv^{G_\bullet}_{G_2}$ are identical.

By the hypothesis of this lemma, G_1 does not subsume G_2, hence there exist strings t_1 and t_2 such that $t_1 \equiv_{G_1} t_2$, but $t_1 \not\equiv_{G_2} t_2$. (Otherwise, G_1 supports some t that G_2 does not; but then t is G_1-equivalent to some $G_\bullet$-supported w, since G_1 is pruned. But t is not G_2-equivalent to w, since t is not supported in G_2. In this case, let $t_1 = t$ and $t_2 = w$.)

Clearly, both t_1 and t_2 are supported in G_1.

Case 1: both t_1 and t_2 are supported in G_2. Since both t_1 and t_2 are supported in both G_1 and G_2, they are both supported in $G_1 \times G_2$. Since they are not equivalent in G_2, they are not equivalent in $G_1 \times G_2$. Let v be the vertex that t_1 and t_2 both go to in G_1, and let v_1 and v_2 ($v_1 \neq v_2$) be the vertices that t_1 and t_2 go to, respectively, in $G_1 \times G_2$. Since G_1 is pruned with respect to $G_\bullet$, there exists a $G_\bullet$-supported string w that goes to v in G_1. If there

[6] For every edge of H that could not be deleted by **Prune**, there is some equivalence class, i.e., vertex, of $G_\bullet$ to which that edge can be associated. Thus the number of edges of H is at most the number of edges in $G_\bullet$, so $|H| < |G_\bullet|$.

do not exist $G_\bullet$-supported strings w_1 and w_2 such that w_1 and w_2 go to v_1 and v_2 in $G_1 \times G_2$, respectively, then Lemma 93 applies, and we conclude that G_1 and $G_1 \times G_2$ are not equivalent on $G_\bullet$-supported strings, a contradiction. Since t_1 and w_1 are equivalent in $G_1 \times G_2$, and since t_2 and w_2 are equivalent in $G_1 \times G_2$, we must have the same equivalences in G_1 (noting that G_1 supports all four strings). But t_1 and t_2 are equivalent in G_1, hence w_1 and w_2 are equivalent in G_1, transitively. But w_1 and w_2 are not equivalent in $G_1 \times G_2$, contradicting our assumption that $G_1 \times G_2$ and G_1 define the same relation on $G_\bullet$-supported strings.

Case 2: at least one of t_1 and t_2 is not supported in G_2. Without loss of generality, assume t_1 is not supported by G_2. Let $t\sigma$ be the shortest prefix of t_1 such that t is G_2-supported, but $t\sigma$ is not G_2-supported.[7]

Both G_1 and G_2 support t, so consider the two paths that t induces in these two graphs; we claim that the $\equiv_{G_1}^{G_\bullet}$ equivalence class containing t is the same as the $\equiv_{G_2}^{G_\bullet}$ equivalence class containing t. Suppose by way of contradiction that this is not the case. Now $t = \sigma_1\sigma_2\cdots\sigma_{|t|}$, so look at the first i such that $\sigma_1\sigma_2\cdots\sigma_i$ is contained in non-identical equivalence classes of $\equiv_{G_1}^{G_\bullet}$ and $\equiv_{G_2}^{G_\bullet}$. Since G_1 and G_2 are pruned, the equivalence class containing $\sigma_1\sigma_2\cdots\sigma_{i-1}$ must contain some $G_\bullet$-supported string w such that $w\sigma_i$ is also $G_\bullet$-supported.[8] Now since the equivalence classes of $\equiv_{G_1}^{G_\bullet}$ and $\equiv_{G_2}^{G_\bullet}$ are identical, the $\equiv_{G_1}^{G_\bullet}$ and $\equiv_{G_2}^{G_\bullet}$ equivalence classes containing $w\sigma_i$ must be identical. But $w\sigma_i$ is both G_1- and G_2-equivalent to $\sigma_1\sigma_2\cdots\sigma_i$, contradicting the assumption that σ_i is the first i where $\sigma_1\sigma_2\cdots\sigma_{|t|}$ reaches non-identical equivalence classes in $\equiv_{G_1}^{G_\bullet}$ and $\equiv_{G_2}^{G_\bullet}$. Hence t is in identical $\equiv_{G_1}^{G_\bullet}$ and $\equiv_{G_2}^{G_\bullet}$ equivalence classes. Now, since $[t]_{G_2}$ contained no outgoing edge labeled σ, no $G_\bullet$-supported string w in $[t]_{G_2}$ is such that $w\sigma$ is $G_\bullet$-supported. But then this means that the outgoing edge from $[t]_{G_1}$ labeled σ can be deleted from G_1, contradicting our assumption that G_1 was pruned with respect to $G_\bullet$. □

7.4 Labeled Equivalence Graphs

Here we consider extending the class of equivalence graphs to allow for labeled vertices. The set of vertex labels is required to possess enough structure to allow computing what will be

[7] Such a prefix exists because the empty string is supported.

[8] If i is 1, w is the empty string. Clearly the empty string must be contained in the equivalence class represented by the root of any equivalence graph, and since the relations $\equiv_{G_1}^{G_\bullet}$ and $\equiv_{G_2}^{G_\bullet}$ are identical, the empty string must be in identical equivalence classes in G_1 and G_2.

the unique most specific generalization (least upper bound) between any pair of vertex labels. Specifically, the structure we require is a finite lattice. The following definition supplies the notion we adopt.

Definition 97 *Let Σ be an alphabet, and let $\mathcal{L} = \langle \Gamma, \bot, \preceq, \sqcup \rangle$ be a finite lattice having partial order $\preceq$ over elements of the set Γ, having (unique) minimum element $\bot$, and having the binary join operator $\sqcup$, that returns the unique least upper bound of its two operands. Then an* $\mathcal{L}$-labeled equivalence graph *over Σ is a graph G that is an equivalence graph over Σ in which each vertex v of G has been labeled with some $\gamma \in \Gamma$.*[9]

For any labeled equivalence graph G and any G-supported string w, let $\ell_G(w)$ denote the label of the vertex reached by w. We now modify the earlier subsumption-induced partial order for equivalence graphs to obtain a partial order on labeled equivalence graphs based on which strings are supported, which strings are equivalent, and which vertex labels are assigned to the supported strings.

Definition 98 *Let G_1 and G_2 be two $\mathcal{L}$-labeled equivalence graphs. Then* G_1 subsumes G_2 *if every G_1-supported string is G_2-supported, if $\ell_{G_2}(w) \preceq \ell_{G_1}(w)$ for every G_1-supported string w, and if every pair of G_1-equivalent strings are G_2-equivalent.*

It is shown in [33] that a CLASSIC description subsumes a second CLASSIC description iff the labeled equivalence graph of the first subsumes that of the second. Thus, positive examples of a labeled equivalence graph G will be graphs G' which are subsumed by G.

We now modify the earlier cross-product definition for equivalence graphs to obtain the cross-product definition for labeled equivalence graphs.

Definition 99 *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two $\mathcal{L}$-labeled equivalence graphs. The* cross-product *of G_1 and G_2, denoted $G_1 \times G_2$, is defined as follows. Let $p_1, p_2, \ldots, p_{|V_1|}$ denote the vertices of G_1 with p_1 denoting the root, and let $q_1, q_2, \ldots, q_{|V_2|}$ denote the vertices of G_2 with q_1 denoting the root. If G_1 or G_2 is empty, then $G_1 \times G_2$ is empty. Otherwise, the vertex set of $G_1 \times G_2$ (a subset of $V_1 \times V_2$) and the edge set of $G_1 \times G_2$ are defined recursively:*

[9] If $\{\mathcal{L}_i\}_{i=1}^m$ is a collection of finite lattices, then the tuples $\{\langle \gamma_1, \ldots, \gamma_m \rangle : \gamma_i \in \Gamma_i\}$ form a finite lattice by taking $\langle \bot_1, \ldots, \bot_m \rangle$ as the minimum element and taking $\langle \gamma_1, \ldots, \gamma_m \rangle \preceq \langle \gamma'_1, \ldots, \gamma'_m \rangle$ exactly when $\gamma_i \preceq_i \gamma'_i$ for each i. We will use this observation later when we return to our discussion of CLASSIC.

- *The graph $G_1 \times G_2$ has a root vertex denoted (p_1, q_1) labeled $\ell_{G_1}(p_1) \sqcup \ell_{G_2}(q_1)$*
- *The graph $G_1 \times G_2$ also has a vertex denoted (p_{i_2}, q_{j_2}) labeled $\ell_{G_1}(p_{i_2}) \sqcup \ell_{G_2}(q_{j_2})$ together with the edge $(p_{i_1}, q_{j_1}) \xrightarrow{\sigma} (p_{i_2}, q_{j_2})$ iff G_1 has edge $p_{i_1} \xrightarrow{\sigma} p_{i_2}$ and G_2 has edge $q_{j_1} \xrightarrow{\sigma} q_{j_2}$ and $G_1 \times G_2$ has the vertex denoted (p_{i_1}, q_{j_1}).*

Note that $G_1 \times G_2$ is an $\mathcal{L}$-labeled equivalence graph whenever both G_1 and G_2 are equivalence graphs.

Theorem 100 *Let $\perp$ be the minimum element of a finite lattice $\mathcal{L}$ having maximum chain length d and having polynomial time computable join operator $\sqcup$, and let Σ be a set of edge labels. Then the algorithm composed of figures 7.3 and 7.4 learns $\mathcal{L}$-labeled equivalence graphs over Σ from membership and equivalence queries in time polynomial in $|\Sigma|$, d, the longest counterexample received, and the size of the target concept.*[10]

Proof: The algorithm is modified only in that the sole vertex of the initial hypothesis is labeled by $\perp$ and the cross-product operator for labeled equivalence graphs is used; specifically, **Prune** does not change – only edge deletions are attempted.

Observe that a counterexample G to H must be of one of the following (not necessarily mutally exclusive) types –

- Some string w supported by both H and G has $\ell_H(w) \prec \ell_G(w)$,
- Some H-supported string is not G-supported, or
- Some pair of H-equivalent strings are not G-equivalent.

If the counterexample G was of either of the latter two types, then G is a counterexample to H based solely on their underlying (non-vertex-labeled) equivalence graphs. Therefore, Lemma 95 applies to show that $\equiv^{G_\bullet}_{\mathbf{Prune}(H \times G)}$ is a proper refinement of $\equiv^{G_\bullet}_{H}$. This can happen at most as many times as there are vertices in $G_\bullet$.

If the counterexample is only of the first type, then the underlying (non-vertex-labeled) equivalence graphs of H and G (and therefore, **Prune**$(H \times G)$) are isomorphic. As such, some

[10] At no point in the run of the algorithm will it have consumed time more than a polynomial in $|\Sigma|$, d, the size of the target concept, and the longest counterexample received to that point.

vertex label of H was generalized. Thus, this first type of counterexample can happen at most $dn_H < d \cdot |H| < d \cdot |G_*|$ times between occurrences of a counterexample of the second or third type, where n_H is the number of vertices in H.

A naïve analysis assumes that in the worst case the label on every vertex must be updated d times between changes to the equivalence classes of the hypothesis. This produces a bound of $O(dn_{G_*}^2)$ on the number of counterexamples received, where n_{G_*} is the number of vertices in G_*.

A more careful analysis recognizes that, until a collection of target equivalence classes were split, generalizing the vertex label for one of the equivalence classes generalized the label for *every* target equivalence class in the collection; thus every target equivalence class need only be isolated once and have its vertex label changed at most d times overall. This bounds the number of counterexamples by $O(dn_{G_*})$. Thus the algorithm witnesses $O(d \cdot |G_*|)$ counterexamples to its equivalence queries.

As in the case of (non-vertex-labeled) equivalence graphs, the number of membership queries used by **Prune** on $H \times G$ is at most $O(|\Sigma| \cdot n_H \cdot n_G)$, where n_H and n_G are the number of vertices in H and G, respectively, so that the number of membership queries used by a single call to **Prune** is $O(|\Sigma| \cdot |G_*| \cdot |\tilde{G}|)$, where $\tilde{G}$ is the largest counterexample yet witnessed by the algorithm.

Finally, since $\sqcup$ can be applied in polynomial time, and since $|H| < |G_*|$ for each hypothesis H, the class of labeled equivalence graphs are polynomial time learnable using membership and equivalence queries. □

7.5 Application to CLASSIC

The graphical representation of CLASSIC used in [36] annotates the vertices of the graph with the AT-LEAST, AT-MOST, FILLS, ONE-OF, and PRIM constraints; thus these constraints make a natural choice for the vertex labels discussed in the previous section. Combining these different kinds of constraints into tuples that serve as the actual vertex label lattice $\mathcal{L}$ is also a natural choice, because ordering tuples $\langle s_1, \ldots, s_k \rangle \preceq \langle t_1, \ldots, t_k \rangle$ exactly when every $s_i \preceq t_i$ is in accordance with the notion of subsumption for CLASSIC [36].

We would like to exploit this similarity to labeled equivalence graphs by employing a prediction preserving reduction using the labeled equivalence graph learning algorithm developed in the previous section. The reduction would use the polynomial time transformations between CLASSIC descriptions and their graphical representations developed by [26, 36] to turn the CLASSIC description examples into a form suitable for **Learn** and to turn the graphical representations queried and hypothesized by **Learn** into CLASSIC descriptions suitable for examination outside of **Learn**. Unfortunately, the semantics imposed on the graphical representation of CLASSIC dissuade us from this black box approach; we will employ the transformations of [26, 36, 33] only after modifying **Learn**.

Because no legal graphical representation of a CLASSIC description has non-attribute roles participating in a SAME-AS constraint, we are prevented from constructing the suggested universal positive example for **Learn**. Thus we are forced to modify **Learn** so that it constructs only labeled equivalence graphs whose edge labels adhere to this restriction on equivalent strings over the role alphabet.

In lieu of a universally positive example that satisfies every constraint of the target, we rely on the semantics of the AT-LEAST and AT-MOST constructors to build a CLASSIC description that always denotes the empty set, which is guaranteed to be a subset (and therefore a positive example) of any target description. Concretely, let r be any role. Then

```
(AND
    (AT-LEAST 1 r)
    (AT-MOST 0 r)
)
```

always denotes the empty set; such a concept is said to be *inconsistent*. Making an equivalence query on the graph of this concept will provide the learner with a positive counterexample, G_0, that satisfies all the constraints of the target. The graph G_0 serves the purpose of the initial universal positive example H used as the initial hypothesis by **Learn**. Having solved the problem of the distinction between attributes and arbitrary roles in CLASSIC, we now address another impediment to a direct application of Theorem 100.

Most of the graphical annotations for CLASSIC possess sufficient structure to apply Theorem 100 immediately. As an example of computing upper bounds for a set of PRIM vertex

label, suppose that we have one label {**red**, **shaggy**, **obese**} and a second label {**brown**, **shaggy**, **graceful**, **obese**} then the set intersection – {**shaggy**, **obese**} – is the most specific label that generalizes both of these labels. Thus, proper generalizations are made by removing some primitive from the set. Consequently, proper generalization can occur at most $O(|\mathcal{P}|)$ times, where $\mathcal{P}$ is the set of primitives. Unfortunately, the AT-LEAST and AT-MOST constraints as used in CLASSIC do not possess the structure required because they do not form finite lattices.

The set of possible AT-LEAST constraints has the ordering (AT-LEAST $k+j$ r) $\preceq$ (AT-LEAST k r) for non-negative integers k and j and role r; thus the AT-LEAST constraints preclude the existence of the minimum element we need to express our initial hypothesis. Note, however, that because the counterexample G_0 obtained above is a positive example that does not always denote the empty set, the AT-LEAST conditions appearing in G_0 finitely bound the AT-LEAST lattice. That is, any AT-LEAST constraint on the vertex of G_0 reached by any target supported string w is more restrictive than the corresponding AT-LEAST constraint reached by w in the target. Therefore, the lattice of plausible AT-LEAST constraints for w is finite, having as the minimal element the value reached by w in G_0.

The set of AT-MOST constraints suffers from a slightly different problem; the most general AT-MOST constraint is the absence of an AT-MOST constraint, which is not an AT-MOST constraint at all. This situation allows an adversary to prevent us from *ever* determining certain targets, even after seeing infinitely many equivalence query counterexamples. To see this consider the target consisting of a root with no AT-MOST constraint for role r. Now consider a sequence of (positive) counterexamples that are simply a root with an explicit AT-MOST condition, where example i asserts (AT-MOST i r). After example i, **Learn** would hypothesize a single root vertex asserting (AT-MOST i r). Clearly, no positive example ever allows the AT-MOST constraint to be removed from the root.[11]

To overcome this difficulty, we use a standard technique of decomposing the concept class to be learned into a collection of subclasses indexed by some parameter m such that the union of these subclasses is the original concept class and such that there is an algorithm that given m learns the subclass indexed by m and that runs in time polynomial in m and the other relevant parameters of the class. This permits us to "guess" efficiently the smallest subclass index in

[11] This problem does not arise in the AT-LEAST constraint because (AT-LEAST 0) is semantically equivalent to imposing no AT-LEAST constraint.

which the chosen target lies - try a value of m and if the algorithm exceeds its polynomial time bound, restart the algorithm using $m + 1$.

The decomposition we use is the value of the largest integer appearing in any AT-LEAST or AT-MOST constraint in any vertex of the graph. Any graph in the subclass indexed by m must have no explicit AT-MOST constraints more general than (AT-MOST m), thus whenever we would infer a hypothesis with an AT-MOST constraint more general than this we simply erase the AT-MOST constraint altogether.

The remainder of the CLASSIC constructs used to annotate vertices of the graph are straightforward. We assume predefined (finite) sets $\mathcal{P}$ of primitives, $\mathcal{I}$ of disjoint primitives, and $\mathcal{R}$ of roles. The most restrictive PRIM annotation demands that individuals satisfy every primitive. In the worst case this annotation would be properly generalized $O(|\mathcal{P}|)$ times before obtaining the null constraint not demanding that individuals satisfy any primitive. The FILLS constraint behaves like the PRIM constraint, except over the disjoint primitives $\mathcal{I}$.

The ONE-OF constraint is most restrictive when no disjoint primitive is specified; as positive examples are seen that name disjoint primitives in a ONE-OF constraint, the named disjoint primitives are unioned to the current collection. The ONE-OF constraint of penultimate generality is the constraint that permits individuals to satisfy any of the disjoint primitives; the most general ONE-OF constraint is simply eliminating the ONE-OF constraint altogether. Thus there are at most $O(|\mathcal{I}|)$ proper generalization made for a ONE-OF constraint.

We are now ready to apply Theorem 100 to CLASSIC.

Theorem 101 CLASSIC *is learnable from membership and equivalence queries in time polynomial in* $|\mathcal{R}|$, $|\mathcal{I}|$, $|\mathcal{P}|$, s, t, *and* m, *where* $\mathcal{R}$ *is the set of roles,* $\mathcal{I}$ *is the set of disjoint primitives,* $\mathcal{P}$ *is the set of primitives,* s *is the number of symbols needed to write the target* CLASSIC *description,* t *is the number of symbols needed to write the largest counterexample description witnessed, and* m *is the largest integer appearing in any AT-LEAST or AT-MOST constraint in the target description.*

Proof: For the vertex labels Γ we chose tuples over the AT-LEAST, AT-MOST, PRIM, FILLS, and ONE-OF constraints - the AND, ALL, and SAME-AS constraints are reflected in the structure of the graph and were handled in Section 7.3. There are $|\mathcal{R}|$ AT-LEAST, $|\mathcal{R}|$ AT-MOST, and $|\mathcal{R}|$ FILLS components in the tuple - one of each for each role in $\mathcal{R}$; there is also

one PRIM and one ONE-OF component in the tuple. Now, ordering the tuples in accordance with the CLASSIC subsumption relation, we find that the longest chain in the subclass indexed by m has length $O(|\mathcal{R}| \cdot m + |\mathcal{R}| \cdot |\mathcal{I}| + |\mathcal{P}| + |\mathcal{I}|)$. Theorem 100 applies to each subclass. We can assume that the largest integer appearing in an AT-LEAST or AT-MOST constraint of the target is i until the time bound under this assumption is exceeded, at which point we restart assuming that $2i$ will upper bound the largest integer appearing in an AT-LEAST or AT-MOST constraint of the target. After at most $\lg m$ restarts, our assumed upper bound will be sufficient and will be at most twice m. This "guessing" produces at most a factor of $O(\lg m)$ slowdown over having known m from the outset. □

This theorem is somewhat unappealing in that it actually demonstrates only a pseudo-polynomial time algorithm for CLASSIC. For any reasonable encoding of the integers the value of m is exponentially larger than the number of digits needed to express it; thus allowing time polynomial in m is allowing us, in a very real sense, time exponential in m. However, observe that the problematic constraints – AT-LEAST and AT-MOST – each form a total ordering. Thus, the pseudo-polynomial running time of the algorithm can be reduced via binary search to polynomial running time. Specifically, **Prune** will, using membership queries, perform independent binary searches for each AT-LEAST constraint of each vertex to minimize the value of each AT-LEAST constraint.

Similarly, **Prune** will determine whether any AT-MOST constraint can be removed from any vertex, and for any constraint that must remain, **Prune** will maximize the integer value used to express the constraint. To determine whether a constraint can be removed, **Prune** tentatively removes the constraint and makes a membership query on the resulting graph, which will remain a positive example if and only if that AT-MOST constraint can be removed. If the result is a negative example, then **Prune** successively doubles the current AT-MOST constraint value and makes a membership query until a negative example is obtained. Then a standard binary search finds the largest integer value for this AT-MOST constraint that produces a positive example. The updated **Prune** is shown in figure 7.5.

The m used in this new version of **Prune** is meant to suggest the index of parameterization used in Theorem 101. This modification reduces the time dependency from a polynomial in m to a polynomial in $\log m$, so that we have the following stronger theorem asserting the fully polynomial time learnability of CLASSIC.

Prune(G)

/* G is a positive example. Each question about whether some alteration to G results in a positive example is answered by asking a membership query. */

1 For each edge e in G

2 If removing[a] e from G produces a positive example, then remove e from G.

3 For each vertex v

4 For each AT-LEAST constraint c of v

5 Let m be the current value for c.

6 Perform a binary search in the range $[0, m]$ to find the minimum value for c for which G remains a positive example, and replace the current value for c with that value.

7 For each AT-MOST constraint c of v

8 If removing c from G results in a positive example, then remove c from G.

9 Else

10 Let m be the larger of 1 and the current value for c

11 While replacing m with $2m$ in c produces a positive example, replace m with $2m$ in c.

12 Perform a binary search in the range $[0, m]$ to find the maximum value for c for which G remains a positive example, and replace the current value for c with that value.

13 Return G

[a] Along with e remove any unreachable component.

Figure 7.5: Updated **Prune**.

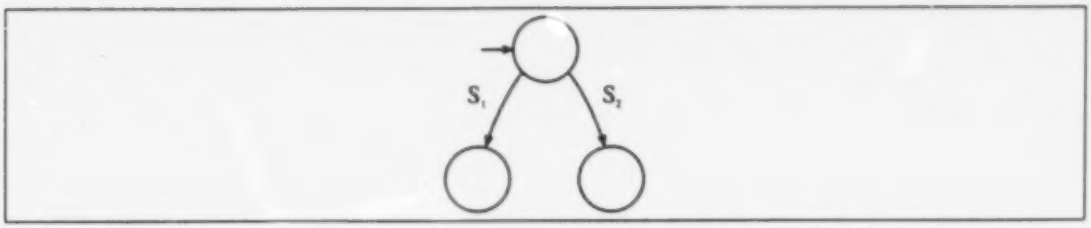

Figure 7.6: A target schema requiring exponentially many membership queries.

Theorem 102 CLASSIC *is learnable from membership and equivalence queries in time polynomial in* $|\mathcal{R}|$, $|\mathcal{I}|$, $|\mathcal{P}|$, s, *and* t, *where* $\mathcal{R}$ *is the set of roles,* $\mathcal{I}$ *is the set of disjoint primitives,* $\mathcal{P}$ *is the set of primitives,* s *is the number of symbols needed to write the target* CLASSIC *description, and* t *is the number of symbols needed to write the largest counterexample description witnessed.*

Proof: The running time of the algorithm depends only polynomially on $\log m$ for the largest integer m explicitly appearing in an AT-LEAST or AT-MOST constraint; this means that the algorithm depends only polynomially on the number of digits needed to represent the largest integer m explicitly appearing in an AT-LEAST or AT-MOST constraint. But the number of digits need to represent this m is certainly no more than s, the number of symbols needed to write the the target CLASSIC description. Note also that the updated version of **Prune** obviates the "guessing" done in Theorem 101 to determine a suitable value of m – we have abandoned the parameterization by m completely. □

7.6 The Insufficiency of Membership Queries

We show in this section that learnability cannot be achieved solely through membership queries. This, coupled with the result of Cohen and Hirsh [33] shows that membership and equivalence queries form a minimal set of queries with which CLASSIC can be exactly learned.

To this end, we now show that it is impossible to learn the class CLASSIC sentences with simple corresponding labeled equivalence graphs in polynomial time using membership queries alone. To do this we exhibit a target for which an adversarial teacher will be able to force the learner to ask exponentially many membership queries before producing a concept equivalent to the target concept.

Theorem 103 *Any algorithm that uses membership queries alone requires $\Omega(2^{|\Sigma|})$ membership queries to distinguish between* CLASSIC *sentences whose equivalence graphs have the form shown in figure 7.6.*

Proof: Figure 7.6 is a template for $O(2^{|\Sigma|})$ distinct concepts based on the partitioning of edge labels[12] of Σ into (disjoint but exhaustive) sets S_1 and S_2. A straightforward adversary argument shows that at least $2^{|\Sigma|-1} - 1$ membership queries are still required:

First, *any* membership query supporting all strings with a single equivalence class is a positive example – no information is obtained by asking such a query. Second, *any* membership query that does not support some edge label from the root is a negative example because a target-supported string is unsupported – membership queries of this form provide no information. Third, *any* membership query that partitions the set of edge labels emanating from the root into more than two sets is a negative example because some pair of target-equivalent strings are inequivalent – such membership queries provide no information in distinguishing among the possible targets.

Thus, only membership queries that partition the set of length 1 strings into two supported equivalence classes can provide any information. Any query that does not partition the edge labels into exactly the same sets as the target is a negative example. There are $2^{|\Sigma|-1}$ such partitions. The adversary simply answers any such query "no" until all but one of the partitionings have been exhausted. Notice that if the learner outputs some conjecture before exhausting all possible partitions, the teacher simply asserts that the conjecture is incorrect by choosing as the target any unexplored partitioning. □

Corollary 104 *The class of unlabeled equivalence graphs (labeled equivalence graphs) (*CLASSIC *descriptions) cannot be learned in polynomial time from membership queries alone, even when the target is known to be acyclic and contain at most three vertices.*

A similar result holds when there are only two roles (i.e., $|\Sigma| = 2$):

Corollary 105 *The class of unlabeled equivalence graphs (labeled equivalence graphs) (*CLASSIC *descriptions) cannot be learned in polynomial time from membership queries alone, even when $|\Sigma| = 2$ and the target is known to be acyclic.*

[12] To adhere to the semantics of CLASSIC, consider a collection of roles all of which are attributes.

Proof Sketch: Simulate a set Σ' of edge labels with $|\Sigma'| = n = 2^k$ by using the labels of Σ as a binary code to label a depth $k-1$ binary tree so that all length $k-1$ strings are supported and every string of length $k-1$ or less is in its own equivalence class. Now construct two more equivalence classes such that every string of length k is in one of these equivalence classes. (See figure 7.7). These last two equivalence classes simulate the non-root equivalence classes of the target in Theorem 103, so that learning this target requires $2^{n-1}-1$ membership queries, but the target has only $\sum_{i=0}^{k-1} 2^i = O(n)$ vertices. □

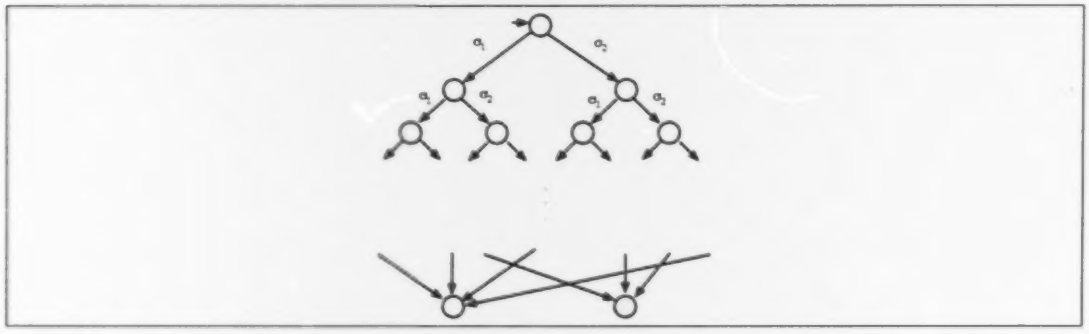

Figure 7.7: This schema requires exponentially many membership queries even though Σ is known to be the set $\{\sigma_1, \sigma_2\}$.

7.7 Membership Query Response Errors

Valiant [99] introduced the notion of PAC learning in the presence of malicious errors [38]. In this learning model, an adversary is allowed to maliciously perturb the probability distribution D on examples by substituting for an example chosen randomly according to D an arbitrary example with arbitrary label; the probability of this substitution occurring is called the malicious error rate [38]. A number of authors have investigated this and related models (for example, see [11, 16, 38, 64, 84, 90, 96, 93]). In this section we investigate the presence of malicious noise in the answers to membership queries and show that very high rates of such noise can be robustly tolerated.

Our model of *persistent malicious classification* errors is as follows. Let $r(\cdot)$ be any polynomial, and let G be any graph. The first time a membership query is made on G the teacher (adversary) flips a coin that with probability $\frac{1}{2} - \frac{1}{r(|G_*|)}$ lands **heads**, where G_* is the target concept. If the coin lands heads, the adversary is permitted to answer the query incorrectly

if he chooses; however, if the coin lands **tails** the adversary must correctly answer the query. Thereafter, the answer to a membership query on G will be the same answer as was first given.

The language of labeled equivalence graphs admits the random construction of a number of independent examples having the property that either all of them are positive examples or all of them are negative examples. Exploiting this property, we give a general test for verifying the answer to a membership query in the presence of even a significant amount of "persistent malicious noise" in the answers to membership queries.

For technical reasons, we consider only *reduced* labeled equivalence graphs – labeled equivalence graphs in which every vertex with maximal label and outdegree 0 has indegree at least 2. Prohibiting maximal-label vertices with indegree 1 and outdegree 0 does not change the ability of labeled equivalence graphs to represent CLASSIC descriptions concisely; however permitting such vertices does illuminate a difference between the semantics of labeled equivalence graphs and the semantics of CLASSIC. A given labeled equivalence graph may be a negative example of the target due to the fact that some target string is unsupported but the vertex reached by this supported string imposes no other semantic constraint in terms of equivalence of strings or vertex label – in CLASSIC such a graph would have arisen from some subexpression stating, "All individuals in the relation r to individuals in this set are individuals in the universe." Clearly such a statement is true for every role r and every individual in the universe; eliminating such a subexpression does not change the semantics of a CLASSIC description, but the two labeled equivalence graphs representing these descriptions would be semantically different. We now claim the following result.

Lemma 106 *Let G_* be the target, let r be any polynomial, let $\delta > 0$ be any probability, and let G be any positive example of G_*. If $|\Sigma| > 1$, then there is a polynomial time algorithm using membership queries with persistent malicious classification error rate $\frac{1}{2} - \frac{1}{r(|G_*|)}$ that determines with probability $1 - \delta$ whether deleting a given edge, e, from G produces a positive example.*

Proof: Let n_{G_*} be the number of vertices in G_*. Let $k = 1 + n_{G_*} + 6r^2(n_{G_*}) \ln \frac{1}{\delta}$. Choose a random string s from Σ^k, and construct a labeled path p from s using k unlabeled vertices where the last symbol of s is the label of a self-loop edge on the last vertex. Redirect the terminus of e to the beginning of this path. If e was deletable, then this new graph is a positive example.

If e was not deletable, then either it must be used to capture some equivalence constraint expressed in the target or it must be used in a path that reaches some vertex label constraint expressed in the target.[13] In either case such a constraint must occur along a path of length at most n_{G_*} within the target, as that is the number of vertices in G_*. However the first n_{G_*} vertices of path p express no constraint of any kind, so that redirecting e must cause some constraint of G_* to be violated. Thus if e is not deletable, this new graph is a negative example.

Now generate $6r(n_{G_*})(r(n_{G_*})-2)\ln\frac{1}{\delta}$ such variations of G being careful not to use the same string s. It is clear that every one of these variations is a positive example if e is deletable and every one of these variations is a negative example if e is not deletable. Ask a membership query on each of these variations. By Chernoff bounds, the probability that fewer than $\frac{1}{2}+\frac{1}{2r(n_{G_*})}$ of these queries are answered correctly is less than δ. Thus, with probability $1-\delta$, e can be deleted if and only if the majority of these queries are answered "yes". □

We hasten to add that the assumption that the edge label set Σ have cardinality at least two is minor; the class of labeled equivalence graphs over an edge label set of size 1 can be easily learned using a straightforward dovetailing algorithm that guesses whether there is a cycle in the graph and if so, how long before the cycle begins. Vertex labels are modified only in response to equivalence query counterexamples and are thus not affected by the errors in membership query responses.

We now have the main result of this section.

Theorem 107 *Let r be any polynomial and let $\delta > 0$ be given. Then the class of reduced labeled equivalence graphs is exactly learnable with probability 1 - δ_0, in time polynomial in* $\ln\frac{1}{\delta_0}$, Σ, d *(the length of the longest chain in the lattice over which the vertex labels are defined), the size of the target concept, and the size of the longest counterexample received, from membership and equivalence queries, even with a persistent malicious classification error rate of* $\frac{1}{2}-\frac{1}{r(G_*)}$.

Proof Sketch: Membership queries are used only in the **Prune** procedure of figure 7.4. Instead of relying on the answers, replace each such query with the procedure in the proof of Lemma 106, with parameter $\delta = \delta_0/s$, where s is the total number of membership queries that the learning algorithm would make without noisy membership queries. Thus, the probability that *any* of the invocations of this procedure is incorrect totals at most δ. If the size of the

[13] This makes use of the assumption that the equivalence graph is reduced.

target concept and longest counterexample to be received is known a priori, then s is given by Theorem 96. If these values are not known, then the standard technique of dynamically guessing s can be employed while increasing the running time only polynomially. □

7.8 Summary

We have demonstrated a positive polynomial time learnability result using membership and equivalence queries for labeled equivalence graphs with vertex labels chosen from a finite lattice, and we adapted this algorithm to obtain a polynomial time algorithm for the "natural" first-order concept class CLASSIC. We then showed that the learnability did not rest solely on the power of the membership queries by giving a non-learnability result for membership query only algorithms. Finally we showed that, although the membership queries are necessary, the accuracy in the answers to those queries need only be polynomially better than 1/2.

Another possible research direction centers around the joining of two graphs. When the join is computed, edge labels are deleted if the label does not appear in both graphs to be joined. What if the edge labels are chosen from a partially ordered set, where joining two edge labels might permit constructing an edge labeled with the least upper bound of the two edge labels to be joined? Because the edge labels capture functions in CLASSIC, such generalizations appear to dwell in world of second order logics. For this reason we believe that a positive result in this direction will rely heavily upon the structure of the underlying partially ordered set.

Chapter 8

Conclusion

The goal of inquiry about automating the knowledge representation process is either to produce a learning algorithm that efficiently automates the encoding of *any* representation that uses some useful representation languages $\mathcal{L}$ or to show that no such learning algorithm is possible. The centerpiece of this thesis was that there do exist learning algorithms for two natural representation languages: propositional Horn sentences and CLASSIC. In addition, this thesis introduced a new method – *consistently ignorant teachers* – of modeling uncertainty in the information being collected. The thesis demonstrated that, by careful consideration of the task at hand, the tools that have been developed in the field of computational learning theory can be used to automate the process of constructing the explanations required by real-world tasks in fields outside computational learning theory. This has been the foundation; it is hoped that by way of example, the potential of adapting computationally learning theory techniques to more ambitious domains has been demonstrated along with the considerations required in making such an adaptation. Adapting the tools of computational learning theory to other representation languages for many other domains lies ahead.

Appendix A

Thesis Synopsis

For expository purposes, we define a variety of characters that the example class might assume. To fully describe an example class we must specify its elements and the way that labels are assigned to elements of the example class by the concept class. Because the examples are intended to teach the learner the behavior of the target, we speak of the examples as being provided by a *teacher*. Thus, we associate a teacher with each kind of example class.

Definition 108 (Standard Teacher) *Let $\mathcal{C}$ be a concept class of propositional formulas, let $\mathcal{X}$ be the set of truth assignments over the variables used in $\mathcal{C}$, and let C_* be the target. A teacher who uses $\mathcal{X}$ and labels examples according to whether C_* is satisfied is called a* standard teacher.

Definition 109 (Entailed Example Teacher) *Let $\mathcal{C}$ be a concept class of propositional (respectively, first-order) logical formulas, let $\mathcal{X}$ be a class of propositional (respectively, first-order) logical formulas, and let C_* be the target. A teacher who uses $\mathcal{X}$ and labels examples according to entailment by C_* (that is, whether every truth assignment satisfying C_* also satisfies the the example) is called an* entailed example teacher.

The next definition provides for a new variety of teacher. The definition allows for teachers who are rational but not omniscient; that is, teachers who have knowledge gaps for which knowledge about the concept class does not help.

Definition 110 (Consistently Ignorant Teacher) *Let $\mathcal{C}$ be a concept class of propositional formulas and let $\mathcal{X}$ be the set of truth assignments over the variables used in $\mathcal{C}$. A* consistently

ignorant teacher *is a standard teacher who is permitted not to know the label of some example x provided that among all the concepts in $\mathcal{C}$ consistent with what the teacher does know, there are concepts C_1 and C_2 which disagree on the label of x.*

In practice, the amount of uncertainty held by a consistently ignorant teacher is measured by assuming that the teacher has some particular subset S of concepts from $\mathcal{C}$ in mind and declares an example to be positive if every concept in S calls the example positive, declares an example to be negative if every concept in S calls the example negative, and declares "I don't know" if some pair of concepts in S disagree about the labeling of the example. The learner is permitted time polynomial in the amount of uncertainty of the teacher.

This work presents several results concerning modeling the world with propositional or first-order concepts. This work also presents alternatives to the idea that an example is a setting of the variables that is labeled according to whether it satisfies the unknown propositional description of the world. Specifically the following results have been obtained.

- Positive Results

  1. Propositional Horn sentences are exactly learnable using equivalence and membership queries given a standard teacher (Theorem 23).
  2. Propositional Horn sentences are exactly learnable using equivalence and membership queries given an entailed example teacher using Horn clauses as examples (Theorem 40).
  3. The first-order class $\mathcal{H}_{2,k}$ of conjunctions of pairs of definite clauses with k-ary predicates and unary functions is exactly learnable using equivalence queries given an entailed example teacher using literals as examples (Theorem 76). Achieving this result produced a new characterization of the regular languages.
  4. Several classes of boolean functions are learnable using equivalence and membership queries given a consistently ignorant teacher (Corollary 63).
  5. Sets of boxes in E^d with samplable intersection are PAC learnable using membership queries and random examples given a consistently ignorant teacher using points as examples (Theorem 66).

6. Embedded multi-symmetric functions are exactly predictable using membership and equivalence queries from a standard teacher (Theorem 5).

7. CLASSIC is exactly learnable using membership queries from an entailed example teacher using CLASSIC descriptions as examples, even when the membership queries are maliciously erroneously answered with probability bounded polynomially below 1/2 (Theorem 107).

- Negative Results

1. PAC learning propositional *2-quasi Horn sentences* (conjunctions of clauses having at most two positive literals) using membership queries from a standard teacher is no easier than PAC learning arbitrary DNF formulas from a standard teacher (Corollary 27).

2. Exactly learning propositional Horn sentences using equivalence queries from an entailed example teacher using Horn sentences as examples is no easier than exactly learning arbitrary DNF formulas using equivalence queries from a standard teacher (Theorem 45).

3. The first-order class $\mathcal{H}_{2,*}$ of conjunctions of pairs of definite clauses is not PAC learnable using random examples given an entailed example teacher using atoms as examples unless NP = RP (Theorem 77).

4. Sets of propositional Horn sentences with samplable intersection are not PAC learnable using membership queries and random examples given a consistently ignorant teacher unless arbitrary DNF are PAC learnable using random examples given a standard teacher, assuming the existence of one-way functions (Corollary 69).

5. CLASSIC is not learnable using only membership queries from an entailed example teacher using CLASSIC descriptions as examples (Theorem 103). As a consequence of this result, random examples and membership queries form a minimal set of queries for PAC learning CLASSIC from an entailed example teacher using CLASSIC descriptions as examples.

Bibliography

[1] D. Angluin. Learning k-term dnf formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Department of Computer Science, Yale University, August 1987.

[2] D. Angluin. Learning propositional Horn sentences with hints. Technical report, Yale University, YALE/DCS/RR-590, 1987.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, November 1987.

[4] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[5] D. Angluin. Requests for hints that return no hints. Technical report, Yale University, YALE/DCS/RR-647, 1988.

[6] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[7] D. Angluin. Computational learning theory: survey and selected bibliography. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 351–369. ACM Press, New York, NY, 1992.

[8] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[9] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40:185–210, 1993.

[10] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proc. of the 23rd Symposium on Theory of Computing*, pages 444–454. ACM Press, New York, NY, 1991.

[11] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[12] D. Angluin and D. K. Slonim. Learning monotone DNF with an incomplete membership oracle. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 139–146. Morgan Kaufmann, San Mateo, CA, 1991.

[13] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, 1979.

[14] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[15] H. Arimura, H. Ishizaka, and T. Shinohara. Polynomial time inference of a subclass of context-free transformations. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 136–143. ACM Press, New York, NY, 1992.

[16] P. Auer. On-line learning of rectangles in noisy environments. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 253–261. ACM Press, New York, NY, 1993.

[17] H. Beck, H. Gala, and S. Navathe. Classification as a query processing technique in the CANDIDE semantic model. In *Data Engineering Conference*, pages 572–581, Los Angeles, CA, 1989.

[18] U. Berggren. Linear time deterministic learning of k-term DNF. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 37–40. ACM Press, New York, NY, 1993.

[19] A. Blum. Separating distribution-free and mistake-bound learning models over the Boolean domain. In *Proc. of the 31st Symposium on the Foundations of Comp. Sci.*, pages 211–218. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[20] A. Blum, P. Chalasani, and J. Jackson. On learning embedded symmetric concepts. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 337–346. ACM Press, New York, NY, 1993.

[21] A. Blum and S. Rudich. Fast learning of k-term DNF with queries. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 382–389, May 1992.

[22] A. Blum and M. Singh. Learning functions of k terms. In *Proc. 3rd Annu. Workshop on Comput. Learning Theory*, pages 144–153. Morgan Kaufmann, San Mateo, CA, 1990.

[23] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[24] Daniel G. Bobrow and Terry Winograd. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1(1):3–46, 1977.

[25] Alex Borgida. Description logics are not just for the `flightless-birds`: a new look at the utility and foundations of description logics. Technical Report DCS-TR-295, Rutgers University Department of Computer Science, 1992.

[26] Alex Borgida and Peter F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. Technical report, AT&T, 1992.

[27] E. Boros, Y. Crama, and P. L. Hammer. Polynomial-time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.

[28] R. J. Brachman, R. E. Fikes, and H. J. Levesque. Krypton: A functional approach to knowledge representation. *IEEE Computer*, 16(10):67–73, 1983.

[29] N. Bshouty. Exact learning via the monotone theory. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 302–311, November 1993.

[30] Z. Chen. Learning unions of two rectangles in the plane with equivalence queries. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 243–252. ACM Press, New York, NY, 1993.

[31] Z. Chen and S. Homer. Fast learning unions of rectangles with queries. Unpublished manuscript, July 1993.

[32] Z. Chen and W. Maass. On-line learning of rectangles. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 16–28. ACM Press, New York, NY, 1992.

[33] W. W. Cohen and H. Hirsh. Learnability of description logics. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 116–127. ACM Press, New York, NY, 1992. A longer version, available from the authors, is in preparation.

[34] William Cohen. Cryptographic limitations on learning one-clause logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Washington, D.C., 1993.

[35] William Cohen. Pac-learning a restricted class of recursive logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Washington, D.C., 1993.

[36] William W. Cohen, Alex Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[37] Mukesh Dalal and David Etherington. Tractable approximate deduction using limited vocabulary. In *CSCSI-92*, Vancouver, May 1992.

[38] S. E. Decatur. Statistical queries and faulty PAC oracles. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 262–268. ACM Press, New York, NY, 1993.

[39] P. Devanbu, R. J. Brachman, P. Selfridge, and B. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 35(5):255–257, May 1991.

[40] Thomas Glen Dietterich, Bob London, Kenneth Clarkson, and Geoff Dromey. Learning and inductive inference. In *The Handbook of Artificial Intelligence, Volume 3*. William Kaufmann, 1982.

[41] Sašo Džeroski, Stephen Muggleton, and Stuart Russell. Pac-learnability of logic programs. In *Proceedings of the 1992 Workshop on Computational Learning Theory*, pages 128–135, New York, 1992. The Association for Computing Machinery.

[42] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 182–194. Morgan Kaufmann, San Mateo, CA, 1988.

[43] M. Frazier and C. D. Page. Learnability of recursive, non-determinate theories: Some basic results and techniques. In *Third International Workshop on Inductive Logic Programming*, 1993.

[44] Michael Frazier, Sally Goldman, Nina Mishra, and Lenny Pitt. Learning from a consistently ignorant teacher. In *Proceeding of the Seventh Annual Conference on Computational Learning Theory*, 1994.

[45] Michael Frazier and Lenny Pitt. CLASSIC learning. In *Proceedings of the seventh annual conference on computational learning theory*, 1994.

[46] Michael Frazier and Leonard Pitt. Learning from entailment: An application to propositional horn sentences. In *Proceedings of the Tenth Internation Conference on Machine Learning*, 1993.

[47] M. Fulk and J. Case, editors. *Proc. 4th Annu. Workshop on Comput. Learning Theory.* Morgan Kaufmann, Rochester, NY, 1990.

[48] M. Fulk and J. Case (editors). *Proc. 3rd Annu. Workshop on Comput. Learning Theory.* Morgan Kaufmann, Inc., August 1990.

[49] P. Goldberg, S. Goldman, and H. Mathias. Learning unions of rectangles with membership and equivalence queries. Technical Report WUCS–93–46, Washington University, Department of Computer Science, November 1993.

[50] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. Exact identification of circuits using fixed points of amplification functions. In *31st Annual Symposium on Foundations of Computer Science*, pages 193–202, October 1990.

[51] Sally A. Goldman and Robert H. Sloan. The difficulty of random attribute noise. Technical Report WUCS-92-25, Washington University, Department of Computer Science, July 1992.

[52] Russell Greiner and Dale Schuurmans. Learning useful Horn approximations. In *Proceedings of the Third International Joint Conference on Knowledge Representation*, Cambridge, MA, October 1992.

[53] D. Haussler. Applying valiant's learning framework to ai concept learning problems. Technical Report UCSC-CRL-87-11, UCSC, 1987.

[54] D. Haussler. Bias, version spaces, and Valiant's learning framework. In *Proceedings of the 4th International Workshop on Machine Learning*, pages 324–336, San Mateo, California, June 1987. Morgan Kaufmann.

[55] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[56] D. Haussler. Generalizing the PAC model: sample size bounds from metric dimension-based uniform convergence results. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 40–45. IEEE Computer Society Press, Los Alamitos, CA, 1989.

[57] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, 1989.

[58] D. Haussler and R. P. Daley (editors). *Proc. 6th Annu. Workshop on Comput. Learning Theory*. ACM Press, July 1992.

[59] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting {0,1} functions on randomly drawn points. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1988.

[60] D. Haussler and L. Pitt, editors. *Proc. 2nd Annu. Workshop on Comput. Learning Theory*. Morgan Kaufmann, San Mateo, CA, 1988.

[61] D. Haussler and L. Pitt(editors). *Proc. 1st Annu. Workshop on Comput. Learning Theory*. Morgan Kaufmann, Inc., August 1988.

[62] Haym Hirsh. Polynomial time learning with version spaces. In *Proceedings of AAAI-92*, 1992.

[63] Henry Kautz and Bart Selman. Speeding inference by acquiring new concepts. In *Proceedings of AAAI-92*, San Jose, July 1992.

[64] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM J. Comput.*, 22:807–837, 1993.

[65] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of Boolean formulae. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 285–294. ACM Press, New York, NY, 1987.

[66] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proc. of the 21st Symposium on Theory of Computing*, pages 433–444. ACM Press, New York, NY, 1989.

[67] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *Proc. of the 31st Symposium on the Foundations of Comp. Sci.*, pages 382–391. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[68] Jorg-Uwe Kietz. Some computational lower bounds for the computational complexity of inductive logic programming. In *1993 European Conference of Machine Learning*, Vienna, Austria, 1993.

[69] Philip D. Laird. Learning from good and bad data. In *Kluwer international series in engineering and computer science*. Kluwer Academic Publishers, Boston, 1988.

[70] C. X. Ling. Logic program synthesis from good examples. In S. H. Muggleton, editor, *Inductive Logic Programming*, pages 113–129. Academic Press, London, 1992.

[71] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[72] N. Littlestone. From on-line to batch learning. In *Proc. 2nd Annu. Workshop on Comput. Learning Theory*, pages 269–284. Morgan Kaufmann, San Mateo, CA, 1989.

[73] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. PhD thesis, Technical Report UCSC-CRL-89-11, University of California Santa Cruz, 1989.

[74] Nicholas Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 147–156, 1991.

[75] P. M. Long and M. K. Warmuth. Composite geometric concepts and polynomial predictability. *Inform. Comput.*, 1993. To appear.

[76] S. Marcus, editor. *Machine Learning*, volume 3/4. December 1989.

[77] E. Mays, C. Apte, J. Griesmer, and J. Kastner. Organizing knowledge in a complex financial domain. *IEEE Expert*, pages 61–70, Fall 1987.

[78] N. Mishra. Private communication, 1992.

[79] T. M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings IJCAI-77*, pages 305–310, Cambridge, Mass., August 1977. International Joint Committee for Artificial Intelligence.

[80] T. M. Mitchell. Generalization as search. *Art. Int.*, 18:203–226, 1982.

[81] S. H. Muggleton. Inductive logic programming. In S. H. Muggleton, editor, *Inductive Logic Programming*, pages 3–27. Academic Press, London, 1992.

[82] C. D. Page and A. M. Frisch. Generalization and learnability: A study of constrained atoms. In S. H. Muggleton, editor, *Inductive Logic Programming*, chapter 2, pages 29–60. Academic Press, London, 1992.

[83] Peter F. Patel-Schneider. A four-valued semantics for terminological logics. *Artificial Intelligence*, 38:319–351, 1989.

[84] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J. ACM*, 35:965–984, 1988.

[85] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, 1988.

[86] L. Pitt and M. K. Warmuth. Prediction preserving reducibility. *J. of Comput. Syst. Sci.*, 41(3):430–467, December 1990. Special issue of the for the *Third Annual Conference of Structure in Complexity Theory* (Washington, DC., June 88).

[87] Vijay Raghavan. Private communication, 1994.

[88] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[89] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Inform. Comput.*, 103(2):299–347, April 1993.

[90] D. Ron and R. Rubinfeld. Learning fallible finite state automata. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 218–227. ACM Press, New York, NY, 1993.

[91] Yasubumi Sakakibara. On learning from queries and counterexamples in the presence of noise. *Information Processing Letters*, To appear.

[92] Bart Selman and Henry Kautz. Knowledge compilation using Horn approximations. In *Proceedings of AAAI-91*, Anaheim, August 1991.

[93] G. Shackelford and D. Volper. Learning k-DNF with noise in the attributes. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 97–103, San Mateo, CA, 1988. published by Morgan Kaufmann.

[94] George Shackelford and Dennis Volper. Learning k-DNF with noise in the attributes. In *Proceedings of the First Annual Workshop on Computational Learning Theory*, pages 97–103. Morgan Kaufmann, August 1988.

[95] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

[96] R. Sloan. Types of noise in data for concept learning. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 91–96. Morgan Kaufmann, San Mateo, CA, 1988.

[97] Robert H. Sloan. Types of noise in data for concept learning. In *Proceedings of the First Annual Workshop on Computational Learning Theory*, pages 91–96. Morgan Kaufmann, August 1988.

[98] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[99] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, vol. 1*, pages 560–566, Los Angeles, California, 1985. International Joint Committee for Artificial Intelligence.

[100] L. G. Valiant and M. K. Warmuth (editors). *Proc. 5th Annu. Workshop on Comput. Learning Theory.* Morgan Kaufmann, Inc., August 1991.

[101] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probab. and its Applications*, 16(2):264–280, 1971.

Vita

Michael Duane Frazier was born October 21, 1963, in North Kansas City, Missouri. His professional experience includes object-oriented expert system shell research and research on recursive query processing in distributed databases, both appointments being at Amoco's Tulsa Research Center. He has also worked with knowledge-based geographic information systems with the U.S. Geological Survey. He received a B.S. in computer science from the University of Missouri – Rolla in 1985, where he was selected as the Outstanding Senior Mathematics Student. He received an M.S. in computer science from the University of Missouri – Rolla in 1987.

In addition to leading teaching assistant orientation seminars for the University of Illinois and receiving both the Teaching Assistant Award for excellence in instruction and the C. W. Gear Award for excellence in graduate research and service, his work at Illinois includes the following papers.

- Learning Conjunctions of Horn Clauses. Invited paper for a special issue of this journal. In *Machine Learning*, volume 9, 1992, with D. Angluin and L. Pitt.
- Learning Conjunctions of Horn Clauses. In *Proceedings of the 31st Symposium on Foundations of Computer Science (FOCS)*, 1990, with D. Angluin and L. Pitt. (This paper is the extended abstract of the preceding paper.) This paper also appeared in *The Third Annual Workshop on Computational Learning Theory (COLT-90)*, 1990.
- Learning from Entailment: An Application to Propositional Horn Sentences. Plenary paper, *The Tenth International Conference on Machine Learning (ML-93)*, 1993, with L. Pitt.

- Learnability in Inductive Logic Programming: Some Basic Results and Techniques. In *The Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 1993, with C. D. Page.

- Learnability of Recursive, Non-determinate Theories: Some Basic Results and Techniques. In *The Third International Workshop on Inductive Logic Programming (ILP-93)*, 1993, with C. D. Page.

- Prefix Grammars: An Alternative Characterization of the Regular Languages. To appear in *Information Processing Letters*, With C. D. Page.

- Learning from a Consistently Ignorant Teacher. To appear in *Proceedings of the Seventh Annual Conference on Computational Learning Theory (COLT-94)*, with S. Goldman, N. Mishra, and L. Pitt.

- CLASSIC Learning. To appear in *Proceedings of the Seventh Annual Conference on Computational Learning Theory (COLT-94)*, with L. Pitt.

- Some New Directions in Computational Learning Theory. To appear in *Proceedings of the First Annual European Conference on Computational Learning Theory*, with L. Pitt.

- Searching with a Non-Constant Number of Lies. Submission to *Algorithmica* withdrawn after an independent publication of the results by A. Pelc appeared.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R- 94-1858	2.	3. Recipient's Accession No.
4. Title and Subtitle Matters Horn and Other Features in the Computational Learning Theory Landscape: The Notion of Membership			5. Report Date April 1994 6.
7. Author(s) Michael Duane Frazier			8. Performing Organization Rept. No. UIUCDCS-R-94-1858
9. Performing Organization Name and Address University of Illinois Department of Computer Science 1304 W. Springfield Avenue Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

Computer automation of tasks is part of the natural progression of encoding information. When the task becomes well understood and repetitive, placing the task under computer control becomes a possibility. Computers were once programmed by rewiring rather than with the use of a modern program, management of a limited memory was once handled by the application programmer rather than by the operating system, and efficient use of the computer's hardware was once obtained by assembly language programmers rather than through a compiler. In other areas, accounting moved from ledger books to spreadsheets, automobile fuel intake left the carburetor for computer-controlled fuel injection, and diagnosis and scheduling left the expert for the expert system.

Current knowledge representation research has sought to provide schemes for encoding knowledge about how a given system behaves, with the goal being accuracy and utility. Can an accurate description be given with the representation language being used? Can the resulting representation be manipulated easily to answer questions about the system being described? To the extent that both questions can be answered affirmatively for some representation language $\mathcal{L}$, encoding information using $\mathcal{L}$ is well understood. Ideally, the goal of encoding knowledge is not the task of encoding, but the product of the encoding task. If such encodings are required for a variety of systems, then question of automating the process of encoding arises.

This thesis considers this automation process to be a question of whether it is possible to automatically learn the encoding based on the behavior of the system to be described. A variety of representation languages $\mathcal{L}$ are considered, as are a variety of means for the learner to acquire a variety of types of data about the system in question. The learning process is abstracted as a *learning problem* in which the goal is to collect efficiently sufficient information to identify some hidden *concept* C represented using the language $\mathcal{L}$. The source of information about C is its relationship to some class of *examples* $\mathcal{X}$ that is assumed to be reasonably available even though C itself is not. In addition to conjecturing guesses as to the identity of C, the learner is permitted ask how C relates to individuals $x \in \mathcal{X}$.

The goal of inquiry about this automation process is either to produce a learning algorithm that efficiently automates the encoding of *any* representation that uses some useful representation language $\mathcal{L}$ or to show that no such learning algorithm is possible. The centerpiece of this thesis is that there do exist learning algorithms for two natural representation languages: propositional Horn sentences and the CLASSIC description logic. In addition, this thesis introduces a new method – *consistently ignorant teachers* – of modeling uncertainty in the information being collected. The goal of this thesis is to demonstrate that, by careful consideration of the task at hand, the tools that have been developed in the field of computational learning theory can be used to automate the process of constructing the explanations required by real-world tasks in fields outside computational learning theory.

17. Key Words and Document Analysis. 17a. Descriptors

PAC Learning, Exact Learning, Inductive Logic Programming, Concept Learning, Membership Queries

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 195
BEST COPY AVAILABLE 188	20. Security Class (This Page) UNCLASSIFIED	22. Price

USCOMM-DC 40329-P71